USENIX
SANS III PROCEEDINGS
The 1994 System Administration, Networking, and Security Conference

The 1994 System Administration, Networking, and Security Conference

SANS III Proceedings

Sponsored by
THE OPEN SYSTEMS CONFERENCE BOARD

with the cooperation of
SAGE
The USENIX Association's Special Technical Group
of
System Administrators

April 4-8, 1994
Washington, D.C.

ISBN 1-880446-59-6

Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

Proceedings of the 1994 SANS III Conference

SANS III

The Third Annual System Administration, Networking, and Security Conference

Sponsored by
THE OPEN SYSTEMS CONFERENCE BOARD
with the cooperation of

SAGE,

The USENIX Association's
Special Technical Group of System Administrators

April 4 - 8, 1994
Washington, DC, USA

Table of Contents

SANS III

The Third Annual System Administration, Networking, and Security Conference

Thursday, April 7

Friday, April 8

PROGRAM REVIEW COMMITTEE

Rob Kolstad, *Berkeley Software Design, Inc.*
Conference Chair

Tom Barrett, *Pacific Bell*
Matt Bishop, *University of California, Davis*
Dave Brillhart, *Harris Semiconductor*
Tom Christiansen, *Consultant and Instructor*
Michele Crabb, *NASA Ames Research Center*
William Howell, *University of North Carolina*
Norman Kincl, *Hewlett Packard*
E. Scott Menter, *Enterprise Systems Management Corp.*
Paul M. Moriarty, *cisco Systems*
Bryan MacDonald, *SRI International*
Alan Paller, *Computer Associates*
Dale Pfaff, *US Naval Research Laboratory*
Marcus Ranum, *Trusted Information Systems*
Elizabeth Zwicky, *SRI International*

A NETWORK PERIMETER WITH SECURE EXTERNAL ACCESS

Frederick M. Avolio

Marcus J. Ranum

Trusted Information Systems, Incorporated
Glenwood, MD

ABSTRACT

A private network that carries sensitive data between local computers requires proper security measures to protect the privacy and integrity of the traffic. When such a network is connected to other networks, or when telephone access is allowed into that network, the remote terminals, phone lines, and other connections become extensions to that private network and must be protected accordingly. In addition, the private network must be protected from outside attacks that could cause loss of information, breakdowns in network integrity, or breaches in security.

While security is important, security measures that are onerous or cumbersome often end up being circumvented by legitimate users of the network in order to get their work done. Because of this, usability — or "user friendliness" — in security features is also of the utmost importance.

Trusted Information Systems, Inc. (TIS) has built a prototype system that provides for strong user authentication, access control, and integrity protection for unclassified but sensitive data on a private (isolated) network (or collection of networks). Furthermore, the prototype system supports the secure connection of the private network to an external internet, as well as dial-up network connections to the private network, via a firewall and secured links, with strong user authentication and encryption of traffic. TIS used a combination of commercial off-the-shelf (COTS) software[1] and custom software for this project.

This paper summarizes the extended system configuration and functional services, and describes the required security services and specific protection mechanisms used to provide these services.

INTRODUCTION

The goal of this work was to take an established and clearly demarked security perimeter and extend it. Extensions could be from the internal network to homes or hotel rooms over phone lines or to users on other hosts on another, outside network, over an external network. The extension of the security perimeter is done for selected services, under well-understood controls, without compromising security.

The initial installation of this work was prototyped for an extended LAN (referred to as the *campus network*) supporting end users who want to be able to do one or more of the following:

- Use a portable computer outside the office in support of their work.
- Exchange electronic mail (e-mail) with users on external networks, such as the Internet.

[1] During the course of this paper, any products or services mentioned by name are mentioned only as examples of existing technologies or systems, and should not be construed as product endorsements.

- Remotely connect from a portable computer or another network into the campus network for access to their files, reading electronic mail, etc.

- Access Internet services, both commercial and non-commercial, from their desktops or while remotely connected from home, a hotel, an airplane, etc.

- Support strong user authentication and privacy in communications.

As a separate task, we were asked to provide a mechanism for examining and verifying incoming mail and routing of validated mail for special handling.

RISKS AND ASSUMPTIONS

On the basis of the needs expressed by the users, TIS drew up the policies and assumptions that would affect how security was implemented. We did not try to quantify the probability of the risks, but included all that seemed possible. We recognized the following risks and make the following assumptions:[2]

- The data we are protecting, while not classified, is highly sensitive and would do damage to the organization and its mission if disclosed or captured.

- The integrity of the campus network directly affects the ability of the organization to accomplish its mission.

- The campus network is physically secure; the people using the campus network are trustworthy.

- Machines on the campus network are considered to be unsecure. We rely on the physical security of the campus to protect them.

- Whenever possible, staff members who are connected from remote sites should be treated as members of the campus network and have access to as many services as is possible without compromising campus security. The security perimeter shall be extended to include them.

- The Internet is assumed to be unsecure; the people using the Internet are assumed to be untrustworthy.

- Staff members are targets for spying; information they carry or communicate is vulnerable to capture.

- Passwords transmitted over outside connections are vulnerable to capture.

- Any data transmitted over outside connections are vulnerable to capture.

- There is no control over e-mail once it leaves the campus; e-mail can be read, tampered with, and spoofed.

- Any direct connection between a campus system (computer) and one on the outside can possibly be compromised and used for intrusion.

- Software bugs exist and may provide intrusion points from the outside into the campus.

---

[2]We purposely use general terms here because these may be applicable to any organization. However, every organization has some different risk concerns and business requirements.

- Password-protected accounts on any campus machine directly reachable from the outside can be compromised and used for intrusion.

- Telephone use by staff members outside the campus is intercepted and recorded. This includes data transmissions (modem connection to the campus).

- Security through obscurity is counter-productive. Easy-to-understand measures are more likely to be sound, and are easier to administer.

POLICIES

On the basis of these assumptions about the environment (both inside and outside) and the risks, we put together a security policy. Its salient points are:

- We are implementing a perimeter defense.

- A tight security perimeter is our main goal. The ability to extend the security perimeter to include staff members at remote sites (at home, traveling, in remote offices) is a close second.

- Security is more important than service; when they cannot be reconciled, security wins.

- The security policy is made to be changed with a change in risks or business (service) needs.

- "That which is not expressly permitted is prohibited. [5]" The security perimeter must be designed to block everything, and services must be enabled on a case-by-case basis only after a careful assessment of need and risk.

- Even if there is a bug in the implementation of a network service, it should not be able to compromise the campus.

- Direct network connections from outside to inside will never be permitted; proxy servers will be used.

- Network services should be implemented with a minimum of features and complexity, allowing thorough and quick review of the source code.

- The completed system should be able to be tested to ensure that it meets security goals.

- Dial-in connections will be controlled through strong user authentication and encryption.
- Passwords will never be transmitted "in the clear"; if they must be transmitted in an unsecured fashion, one-time passwords will be used.
- Official communications via e-mail will contain digital signatures for authentication of the sender and non-repudiation.
- An individual user should be able to use the same mechanism for user authentication across all services that require it.
- Data on mobile computers will be encrypted.

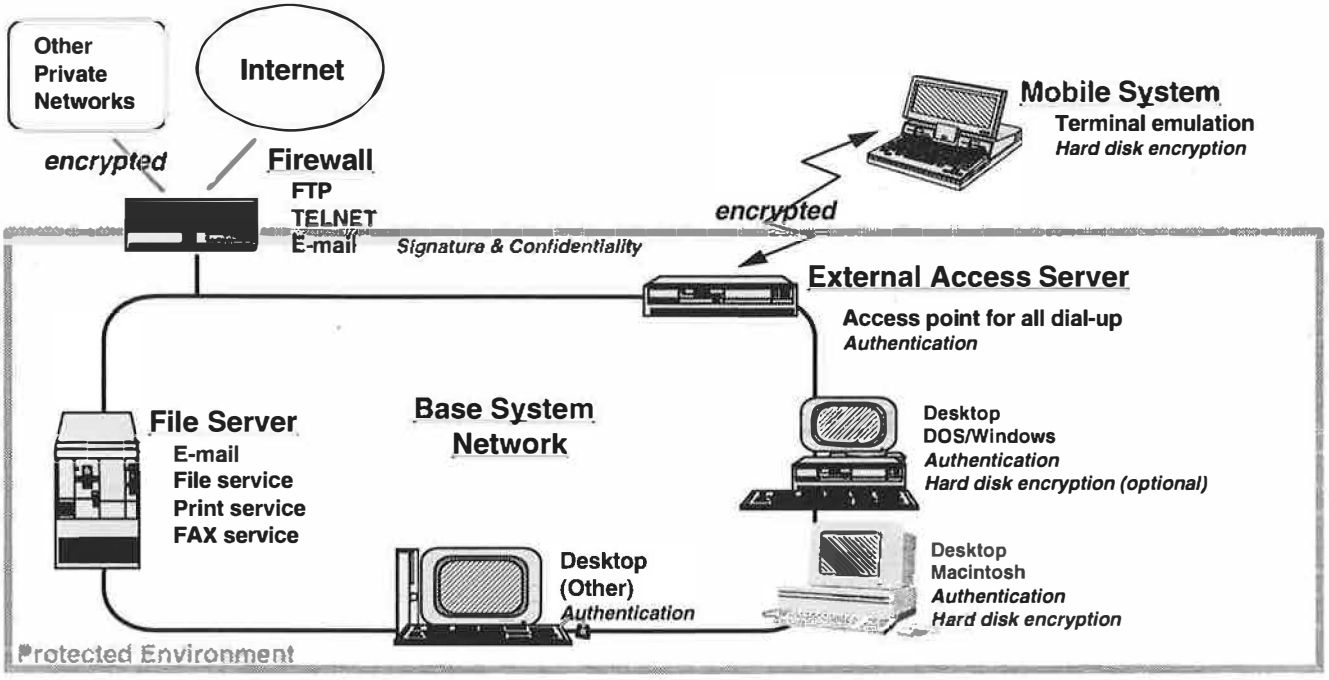


Figure 1: System overview

SYSTEM OVERVIEW

The prototype system provides protection for the campus network and its Internet and dial-up connections. This work forms a model that can be used for secure extensions of other private networks. A primary goal of this work was to provide secure and easy-to-use remote access to the protected network, providing protection against accidental or malicious modification or disclosure of data, and providing strong user authentication. The system configuration appears in Figure 1.

The prototype system incorporates the following protection mechanisms to support our goal of being able to extend the security perimeter.

- A strong user authentication service to establish user identity

- File integrity and protection on desktop and portable computers, via disk (or file system) encryption with user authentication mechanisms

- E-mail security using Privacy Enhanced Mail [1] software to provide a mechanism for examining and verifying incoming mail, based on valid digital signatures, and for identifying validated mail for special handling

- Encryption of all communication between the campus and remote users, including link encryption for dial-up connections

- An Internet firewall toolkit to enforce the security perimeter and provide secure Internet connection for e-mail, FTP, TELNET, and other services

This paper discusses each of these protection mechanisms as employed in the prototype system.

USER AUTHENTICATION

We use "authentication" as defined by the National Computer Security Center's "Red Book" [2] as "(1) to establish the validity of a claimed identity or (2) to provide protection against fraudulent transactions by establishing the validity of ... the individual ...." Identification of a user is often accomplished on computers through the use of a user name and password pair. The password is kept secret and must be difficult to guess; only the user knows the proper name and password pair to use. In reality, passwords are often weak (guessable). Further, in the case of identifying users over outside communication links, there exist opportunities for capture of the user name and password information (although the password is usually not echoed, it is transmitted over the communications link "in the clear"). Consequently, while it would seem that a user name and password pair constitute good identification criteria, the password is too easily guessed or captured. In the prototype system, authentication of a user is done in such a fashion that we can apply a high degree of trust to the identification. This can be accomplished with one-time passwords, or authentication devices or tokens such as Digital Pathways SecureNet or Security Dynamics SecurID. We use all three mechanisms, as examples, to show different ways that strong user authentication can be done.

One-time passwords are passwords that are used only once. A user in a trusted or protected environment, before leaving for a remote location, will identify himself to the computer and generate a list of passwords or phrases to use as part of a challenge response system. The list generated by the computer gives sets of word pairs or numbers and a set of words associated with those numbers [3].

Challenge	Response
coddle	slugfest
toaster	require
doghouse	skateboard
eagle-eye	strummer

Figure 2: Word pairs

Challenge	Response
22	want dew hurl wavy otter stop
23	wise gal miss be king ball
24	iris a gap lure now red
25	stun otiose tom too oven glow

Figure 3: Number and Word List Combinations

If a user was to login and gets "doghouse" in response to the user name entered, the user would have to enter "skateboard" in response (see Figure 2). This would validate the user to the system. It would not matter if someone captured that data on the communications line because the challenge "doghouse" would never be given again with the expected response of "skateboard." A more elaborate system is illustrated by the second table (Figure 3). A publicly available program called S/Key works in this fashion.[3]

The benefit of a word list method such as S/Key is that it is easy to use, it is flexible, and it is inexpensive to implement. Before a user leaves the campus, a list of challenges and responses can be generated, or the user can rely on mobile computer-based software to generate responses.[4] S/Key generates them based on a secret password the user shares with S/Key on a server on the campus network. No challenge will be issued more than once; even if someone was to capture the remote user's challenge and response, it would be unusable. There is a risk of compromise, however, if someone gets hold of the list, knows what it is for, and knows the associated login name.

We recommend the mobile PC-based approach to response generation, if using this method, in which the response is generated only after the user types in his password to the program. While this method is not as secure as the next two we will discuss, for many organizations it will be "good enough" based on their security policies.

User authentication with systems like the Digital Pathways requires the user to have a hand-held device about the size of a pocket calculator. The user identifies himself to the card using a personal identification number known only to the user. The user logs in using his login name and is given a numeric challenge. The user then keys the challenge into his encryption device and reads the response his device displays. This is then typed in as the response to the challenge. The software on the host knows the seed used for encryption on the handheld device issued to the user. The host does the same calculations as the device does. Since the challenge is random, the response given is of no use to anyone eavesdropping on the connection. Practically speaking, that particular numeric challenge will never be given to that particular user again. If the "password" is compromised, it is not a security threat, since the "password" is never reused.

SecurID, from Security Dynamics, operates in a similar manner. Each SecurID card has a unique seed used for encryption. Every 30 or 60 seconds (depending on the model used), the card shows a different numeric value, based on the seed and the date and time. The server software applies the same algorithm, based on the login name used (and a table indicating which card is issued to that user). This removes the challenge step from the authentication process, but otherwise is similar to other methods of user authentication. A personal identification number is used on this device also. While SecurID has versions

[3]A one-time password program using word pairs is in use at the Massachusetts Institute of Technology. S/Key, a publicly available package from Bellcore, implements this second example of a numeric challenge and a word list response.

[4]Mac and DOS software exist.

of their device that do not require entering a PIN, we recommend user identification to the card to avoid the risk mentioned with S/Key (above). The vulnerability with SecurID is that there is a fixed time period when the same response would work. This is easily fixed in software, allowing only 1 login per period. We used both Digital Pathways cards and SecurID cards in our implementation. Other products could be used as well.

Access to the outside from the inside for TELNET or FTP can require user authentication. Since the user is already authenticated in this manner on the inside network, we made a policy decision not to re-authenticate if the connection originates on the inside. All access to the internal network by users on the outside — via TELNET, FTP, or modem — requires user authentication. In this way, we can trust that the remote user is who he or she claims to be.

Access for "dial-in users" is provided by an authentication server that supports S/Key, SecurID, and SecureNet authentication of the remote users. Once connected and authenticated, a remote user will be considered a member of the campus network and will have full access to all services provided. The authentication server runs on a general-purpose host holding the user authentication data bases and is available as a network service (as described in the firewall toolkit section, later).

The network authentication server provides a generic authentication service for network applications. The authentication server must run on a secure host, since its database could be a point of attack. It can embed support for multiple forms of authentication systems simultaneously.

FILE INTEGRITY AND PROTECTION

Through the use of encryption and access control based on strong user authentication (the same user authentication described above), computer and file access may be limited and controlled. Desktop computers on the campus can be configured to be privately owned or general-use computers. Any individual with an account on the campus network is able to use any general-purpose desktop computer to access his or her personal files on the network. Users authenticate themselves to the network servers by using authentication technology through an authentication server, and so access to personal files is permitted. Unauthorized access is prohibited. Whether this form of file protection is used varies with the workstation and the office it is in. The assumption that the campus was a protected environment allowed for looser protection on computers within the environment and, so, fits the security policy.

Personal desktop computers can be configured such that only the "owner" has access to the computer. In this case, commercial off the shelf software (such as Watchdog or User-EZ) is used to encrypt the local disk based on a password or authentication device identifier. Only the authorized user of the computer can gain access to the local files, because only with the proper password or response to a challenge (again, as with user authentication above) can the disk be decrypted. As with network file servers, the policies permit us to forgo strong user authentication in this case.

Given the risks, assumptions, and user requirements, portable personal computers were a special security concern. If the notebook computer is used for accessing and working with sensitive data, there must be a way to protect that data while on the portable computer. The portable computer, even when disconnected from the campus network, must still be considered an extension to the network and protected accordingly. It is too easy to leave a notebook PC behind in a taxi or plane, or to have someone examine it if it is left behind in a hotel room.

Therefore, notebook computers are also protected with disk encryption. To gain access to the notebook computer, the user will have to authenticate herself to the unit using a token (e.g., SecurID) or a password. The notebook computer's disk is encrypted and unusable without valid authentication. Risks associated with the use of such software, besides the implementation being dependent on proper use, are the problems of "instant off/on" modes on portable computers (bypassing the initial "login" sequence), and

the computer being stolen while powered up. All software we checked had a time-out option, requiring periodic re-authentication of the user if the system is idle.

SECURE ELECTRONIC MAIL

To secure e-mail, additional security measures are desirable, especially for official business. Privacy Enhanced Mail (PEM) [4] provides three essential features necessary for official business:

- Integrity: Any electronic data received via PEM can be shown to be the same data that was sent. In other words, it can be proven that what the reader is reading is what the sender sent; no one changed the data in transit. This doesn't prevent tampering, but it does detect it.

- Authentication and non-repudiation: The sender, using PEM, digitally signs the e-mail with a digital signature only the sender can use. Digitally signed e-mail can be checked for a valid and true signature, just as the messages can be checked for integrity. A digital signature, then, provides for authentication of the sender information on e-mail, as well as non-repudiation by the sender (it can be proven that only the sender — or someone with access to the sender's private encryption key — could have sent the message).

- Privacy: Using any one of various encryption algorithms available, e-mail can be sent from one user to another, such that only the intended reader can decrypt the message.

In addition to using these features on individual mail messages between users, a PEM-based gateway and router is used on a mail gateway or hub.[5] This mail hub software can be configured to selectively sign or encrypt outgoing messages. This provides automation of these services between networks or from the campus network to a set of outside user mailboxes. This is currently in use at TIS.

The mail hub software also can examine incoming e-mail for validly signed electronic mail. If a valid digital signature is found, this signature is checked against a list of signatures for "smart routing" of the mail. In an organization that receives a large number of electronic messages for an individual, such as is the case for e-mail for a marketing department or high level executive, it is desirable for the computer system to make some mail routing decisions before human intervention is needed. Some mail might go to general handlers, such as is the case for the majority of mail received by a marketing department. Mail from certain individuals, however, might need to receive special handling or go directly to an individual recipient. For example, mail addressed to a special recipient with a digital signature indicating that it is from an individual on a list of high-priority senders gets different handling than unsolicited e-mail addressed to the same mailbox. A PEM-based gateway and router is able to do this automatic checking and routing.

Finally, this same mechanism could be used for digitally signing official publications sent by e-mail to recipients or posted to Usenet. Since it is trivial to fake e-mail and Usenet postings, there is a significant benefit to e-mail and Usenet postings that can be checked for integrity and authenticity. Only valid digitally signed publications, as an example, would be released to Usenet where they can be independently verified.

ENCRYPTED REMOTE COMMUNICATIONS

Protection of remote connections to the campus network is provided in four distinct ways:

---

[5] A *mail gateway* is a computer system that sits between one environment and another and handles the relay of mail. A *mail hub* is a system that acts as a relay site from gateway systems and other mail hubs.

- TELNET connection from an Internet site: Special encrypting software for an encrypted TELNET session was written. The TELNET session is encrypted from a special TELNET client to a special TELNET server.[6] This allows extending the security perimeter to include the remote user and terminal (although it does not include the remote network). This software will evolve to match Internet standards as they develop.

- Dial-up connection via modem and terminal emulation software: This uses an encrypting terminal emulator and server or encrypting modems. This could also use a cellular telephone connection. The terminal emulator on the remote system (a portable computer, for example) encrypts its end of the session. At the private network side of the connection, a decrypting process runs on an external access server. Because the entire link is encrypted, the security perimeter is extended to include the remote user and terminal. We build this software by modifying a version of Kermit to do encryption.

- Dial-up connection for Serial Line Internet Protocol (SLIP) or Point-to-point Protocol (PPP): This provides an encrypted TCP/IP session, using software or encrypting modems. If the software encryption is employed, this could also be done over a cellular telephone link.[7] There is a risk with allowing this. With this method, there is no way to prevent a user from connecting a whole external network to the campus via SLIP. A user can connect his mobile computer to a network and then, via modem, connect to the campus network. If the mobile computer software allows IP forwarding (allowing the computer to act as a network router), the campus network would be compromised, lowering the level of security to the level of security on the remote, unknown network. Because of this, we recommend caution with allowing this service. Encrypted SLIP was written but will evolve to support any Internet standards developed.

- Point-to-point encryption: Encrypting routers or software allow for encryption of all IP packets between pairs of networks. The benefit of an encrypted link, beyond the obvious observation that no data is ever sent "in the clear" over untrusted paths, is that it allows the joining of network "islands." Because of this, users and hosts on a remote network can be treated as local, trusted users. The security perimeter can be extended to include communication packets over the outside network and the entire remote network. Note that while this is permitted, this should only be allowed if both sites share a common security plan or profile. There are commercial products to support this. We used UUNET Technologies' LAN Guardian in our prototype.

SECURE INTERNET CONNECTION: THE FIREWALL TOOLKIT

The connection to the Internet is through a filtering router in conjunction with a security and applications server. This server runs a version of the UNIX operating system and provides e-mail and DNS service, as well as application support. There are commercial Internet Firewall products that support such a configuration, including those from ANS, Digital Equipment Corporation, and Raptor. As part of this project TIS developed an Internet firewall toolkit consisting of software modules and configuration guidelines, to provide a publicly available base for "industrial strength" firewall security for organizations who desire to build their own firewalls.[8]

---

[6] TELNET encryption will use the Internet standard when available.

[7] At this writing, we do not know of any commercial modems that support both cellular *and* encrypted communications.

[8] The TIS Firewall Toolkit is available in source form via anonymous ftp from `ftp.tis.com: /pub/firewall/toolkit/fwtk.tar.Z`.

Figure 4. An Internet Firewall

The rationale for installing a firewall is almost always to protect a private network against intrusion. In most cases, the purpose of the firewall is to prevent unauthorized users from accessing computing resources on a private network, and often to prevent unnoticed and unauthorized export of proprietary information. In some cases, export of information is not considered important, but in many cases this is a major, though possibly unwarranted, concern. Many organizations will want to address the problem by not connecting to the Internet at all. This policy can be difficult to enforce. If the private network is loosely administered or decentralized, a single enterprising individual with a high speed dial-up modem can quickly arrange an Internet SLIP connection that can compromise the security of an entire network [5].

The purpose of an Internet firewall is to provide a single point of defense with controlled and audited access to services, both from within and without an organization's private network. Internet firewalls tend to be implemented in one of three ways: either (1) with the security engineered on one or more hosts, (2) with routing between the private network and the Internet (or any two networks) blocked, or (3) via screening rules in a commercial router, with direct routing between the Internet and the private network. This design decision sets the general stance of the firewall, favoring either a higher degree of service or a higher degree of isolation. In either case, it is sometimes desirable to support proxy forwarders [5] on the firewall, to act as a gateway for specific applications such as FTP or the X Window System. A proxy forwarder for a network protocol is an application that sits on a firewall host and connects specific service requests on one side of the firewall with servers on the other side, in a controlled, auditable, selective, and secure fashion, and often gives the illusion to the software on both sides of a direct point-to-point connection.

The TIS Firewall Toolkit is designed to be used with a host-based security policy, but its components can be used with router-based firewalls. In this paper, we will focus on the former. In a host-based firewall, the security of the host is crucial; once it is compromised the entire network is often open to attack. Still, we believe that a host-based firewall is superior to other designs because of the ease with which it can be maintained, configured, customized and audited. The TIS Firewall Toolkit is designed to be used in conjunction with router-based screening as extra security. To minimize risks, the services that are provided on the external machine ("bastion host") [5] are sharply curtailed and each service is subjected to review. On the "standard" firewall configuration, the only services supported are DNS, SMTP, FTP, NNTP, TELNET (via proxy and forwarding servers), and user authentication. Other proxies such as Treese's X Window System proxy [6] can be added to this architecture.

The firewall toolkit functionality can be broken down into 6 areas: logging, electronic mail, the Domain Name Service, FTP, TELNET, and TCP access control.

Logging

Through the `syslog` facility, all significant security events are logged to a protected host on the internal network. The version of `syslogd` that the toolkit uses is based on the BSD "net2" sources, with some modifications to support pattern-matching and program execution on matched patterns. Many systems administrators have `cron` jobs set up on their systems to alert them of possible security problems by searching the system logs at regular intervals. By permitting the systems manager to easily add regular expressions to the `syslogd` configuration, security-related log messages can be identified instantly, before log files can be tampered with. `Syslogd` contains further modifications that permit an arbitrary command

to be invoked with any specified logging rule, so that, for example, vitally important security log events can be delivered to the systems manager's beeper, or delivered immediately by electronic mail.

Electronic Mail

Mailers are one of the favorite points of attack against UNIX systems. The Morris Internet worm exploited a well-known hole in the standard UNIX SMTP server, `sendmail`. Many systems running sendmail, including those with Internet firewalls, were penetrated by the worm. A few that had replaced sendmail with other SMTP servers were not [7]. Typically, the problem with mailers is twofold: they are complex and perform file system activity, and they often require privileges so that they can manipulate users' mailboxes.

To secure mail service, direct network access to sendmail is prevented. A simple program that implements a skeleton of the SMTP protocol is presented on the SMTP port on the mail server. This sendmail-proxy, called *smap*, is small enough to be subjected to a code review for correctness (unlike sendmail) and simply accepts all incoming messages and writes them to disk in a spool area. Rather than running with permissions, the sendmail-proxy runs with a restricted set of permissions and runs "chrooted"[9] to the spool area. A second process is responsible for scanning the spool area and delivering the mail messages to the real sendmail for delivery — a mode of operation in which sendmail does not require permissions for operation. Many Internet firewalls run sendmail and rely on "trustworthy" versions of the software; running the mail software in a reduced-permissions mode is a more general solution to the problem, neatly side-stepping the issue of whether or not a given version of sendmail contains bugs.

While smap answers all valid sendmail SMTP commands sent to it, it does not execute any of them except those directly involved with mail exchange: HELO, FROM, RCPT, DATA, and QUIT. Smap preserves sendmail's functionality, while preventing an arbitrary user on the network from communicating directly with sendmail. Analyzing the sendmail program's 20,000 lines of source code for bugs is a sizable task when compared to analyzing smap's 700 lines.

Domain Name Service (DNS)

The name service software available for UNIX implements an in-memory read-only database. As such, it cannot be used to gain unauthorized access to a system. Past attacks on firewalls have used name service spoofing as a technique for impersonating trusted network hosts. In order to remove the threat of name service spoofing, the firewall does not rely on name service for any security related information. The name server software is necessary for high performance large-scale mail systems and is configured so that the only application that relies on name service for addressing is the electronic mail system.

FTP

The FTP application gateway is a single process that mediates FTP connections between two networks. Since it performs no disk access other than reading its configuration file and is a small and relatively uncomplicated program, it can be proven that it is not capable of compromising the security of the system. Just to be certain, the application gateway runs as a non-privileged user, after being "chrooted" to a private directory on the system. To control FTP access, the application gateway reads a configuration file, containing a list of FTP commands that should be logged, and a description of what systems are allowed to engage in FTP traffic. All traffic can be logged and summarized. Optionally, the gateway can permit

[9]Chroot is a mechanism in UNIX whereby a process is irrevocably confined to a single branch of a filesystem. Once the chroot is performed on a process, the restricted branch of the filesystem is treated as its root directory. This mechanism makes it easy to prevent access to device files or files such as the password file.

FTP traffic from the Internet to the campus network for users who first authenticate themselves to the system.

TELNET

The TELNET application gateway is a small, simple application that mediates TELNET traffic. As with the FTP application gateway, the only file accessed is the configuration file that is read at start-up. Immediately after the configuration file is read, the TELNET application gateway is "chrooted" to a restricted directory, where it runs as a non-privileged process. The TELNET gateway's configuration file allows specification of which systems or networks can use it, and what systems or networks it will permit connection to. Initially, it will be configured to permit campus systems to use the gateway to connect to Internet systems, but not vice-versa. Optionally, the TELNET gateway can require strong authentication before permitting use. All connections and their durations are logged.

TCP Access and Use

On BSD-based UNIX systems, most network processes are started up by an initial connection to a general-purpose network listener `inetd`, which establishes a connection between the incoming request and the program to service the request. For example, an incoming request for the TELNET service is "heard" by the running network listener. The program, according to `inetd`'s configuration file and the entry for TELNET, is executed and connected to the incoming request.

`Inetd`, the internet services daemon, performs no function other than to invoke specified processes to manage network services when a system attempts to connect to them. Some vendor implementations permit a systems administrator to specify the user-id that the service should be invoked as, but there is no provision for limiting access based on the source of the request. A variety of implementations of "wrapper" processes is available on the Internet [8] with varying functionality.

The toolkit uses a "wrapper" process called `netacl`, which provides support for all TCP-based services. (If only TCP-based services are supported, UDP services are disabled and are no longer a threat worth worrying about.) `Netacl` has no great advantages over other versions of TCP wrappers, other than its minimal size (240 lines of code, including a large copyright header and comments), its lack of support for UDP (purposely), and its sharing a common configuration mechanism with the other tools in the toolkit. We believe that these differences provide a significant security advantage.

TCP Plug-Board Connection Server

Certain services such as Usenet news are often provided through a firewall. In such a situation, the administrator has the choice of either running the service on the firewall machine itself or installing a proxy server. Since running news, for example, on the firewall exposes the system to any bugs in the news software, it is safer to use a proxy to gateway the service onto a "safe" system on the campus network. Plug-gw is a general purpose proxy that "plugs" two services together transparently. Its primary use is for supporting Usenet news, but it can be employed as a general-purpose proxy if desired. Plug-gw is configurable, as are the other proxy servers. Since it only acts as a data pipe, it performs no local disk I/O and invokes no subshells or processes. Like the other proxy servers, it logs all transactions.

UDP

Since we decided that no direct traffic would be permitted between an outside system and an inside system, and since UDP is connectionless and point-to-point (and so cannot be used through network proxies), UDP services are not allowed.

User Authentication

The network authentication server authd provides a generic authentication service for network applications. Its use is optional, required only if the firewall FTP and TELNET proxies are configured to require authentication. Authd's purpose is to provide a generic interface to multiple forms of authentication. For large organizations, where several forms of authentication challenge/response cards are in use, authd can link them all together to use a single database. A simple administrative shell is included that permits the authentication database to be manipulated over a network, with optional support for encryption of authentication transactions. The authd database supports a basic form of group management; one or more users can be identified as the administrator of a group of users, and can add, delete, enable, or disable users within that group. Authd internally maintains information about the last time a user authenticated to the server and how many failed attempts have been made. It can automatically disable accounts that have multiple failures. Extensive logs are maintained of all authd transactions. Authd is intended to run on a secured host, such as the bastion host, since its database is a possible point of attack.

OBSERVATIONS

Securing a network's perimeter is an interesting exercise in tradeoffs. Since many useful tools (e.g., encrypting terminal emulators and PPP servers that use specific authentication systems) are not available in off-the-shelf form, a certain amount of software development is required. A whole range of interoperability problems is encountered and must be overcome. Hard choices must be made as to whether to solve problems by replacing code, fiddling with configurations, or persuading users to change their habits. Often none of these choices is appealing.

Implementing security-related software designed to resist intrusion requires organized thought if one wishes to have any confidence in the security of the result. During implementation, several design principles emerged:

- Statement of Mission. Most importantly, one should draw up a clear statement of mission before beginning to design one's solution. Assumptions should be stated, and design goals should be identified. This includes formulating a list of risks to defend against, ranging from things that absolutely cannot be permitted to happen, down to things that are interesting but can be ignored. Each identified risk should have a solution proposed for it, even if the solution is "We'll ignore that one." Risk analysis often brings to light issues that are easy to overlook. For each solution that is proposed, a means of verifying the solution's correctness should also be presented; otherwise it's hard to tell if one has protected oneself or simply muddied the waters. Problems must be addressed at a global level. Deciding "What am I trying to protect and protect against?" or "What am I trying to do?" produces a better solution than starting with the components: "How do I secure my Internet link? How do I secure my dial-in lines?"

- Keep It Simple, Stupid (KISS). If the source code for a security-related program is large enough and complex enough that it cannot be checked over in a couple of minutes, it's too complicated to be secure. There are many packages for UNIX systems that control security-related information that are entirely too feature-laden to be maintainable or trustworthy. Also, if you have a reasonable overall goal, it is easier to articulate, implement, and administer.

- Assurance. It should not be comforting to trust your network security to software that has been "hammered on enough that there aren't likely to be any more bugs." Employ configuration practices such that, even if your software has bugs, an attacker cannot use it to compromise the system. An example of this approach is causing the SMTP listener to run "chrooted," without permissions, so that even if someone manages to find a hole in it, they are trapped in an isolated compartment on the system.

- Minimize Risk. If a network service is disabled, it can't hurt you. Your security is better if you shut everything off and turn it on bit by bit than if you try to run around and shut off only the services that are known to be dangerous. This means, in a nutshell, that you must accept that what you don't know *can* hurt you. The other alternative leaves you in an "arms race" against your potential attackers.

- Verify. When you postulate that you have shut down all unnecessary network services, have a procedure in place to verify that, in fact, it is the case. While this may seem like a nebulous task, it is really fairly easy. In keeping with a simple solution, acceptable failure modes are defined and a means of assuring that those failure modes will be met is developed.

While this project was motivated by the requirements of the Executive Office of the President, the work we have done can be used by any organization that wishes to secure their external computer communications. We have provided security with easy-to-use remote access to and from a protected network, while protecting data from disclosure, in an environment using strong user authentication. Further, many of these methods can be employed in an environment that already has mechanisms in place for addressing some of these concerns. Others, such as the Firewall Toolkit, can be used independently of the other methods discussed in this paper, but should only be done in conjunction with a strong security policy, which has thoroughly examined security threats and concerns and mapped out other counter-measures.

ACKNOWLEDGMENTS

This work was done under a contract from the U. S. Department of Defense, Advanced Research Projects Agency (ARPA), number DABT 63-92-C-0020.

REFERENCES

1. James M. Galvin and David M. Balenson, "Security Aspects of a UNIX PEM Implementation," Proceedings of the 3rd USENIX UNIX Security Symposium, September 1992.

2. National Computer Security Center, "Trusted Network Interpretation of The Trusted Computer System Evaluation Criteria," July 31, 1987.

3. Phil Karn, Neil M. Haller, and John S. Walden, Bellcore, S/Key software kit, available via anonymous ftp from `thumper.bellcore.com` `/pub/nmh/skey/*`.

4. Stephen T. Kent, "Internet Privacy Enhanced Mail," Communications of the ACM, August, 1993.

5. Marcus J. Ranum, "Thinking About Firewalls," Proceedings of Second International Conference on Systems and Network Security and Management (SANS-II), April, 1993

6. Winfield Treese and Alec Wolman, "X Through the Firewall, and Other Application Relays," Cambridge Research Lab Technical Report 93/10, Digital Equipment Corporation, May 3, 1993.

7. Bill Cheswick, "The Design Of a Secure Internet Gateway," Proceedings of the 3rd USENIX Security Symposium, September 1992.

8. Wietse Venema, Department of Math and Computing Sciences, Eindhoven University of Technology, The Netherlands. USENET Archives, `comp.sources.misc`, `Volume 20`, August 1991.

Who's Trusting Whom?
How To Audit and Manage Users' .rhosts Files

Michele D. Crabb (crabb@nas.nasa.gov)
- Sterling Software / NASA Ames Research Center

ABSTRACT

The use of *.rhosts* files to provide trust between systems has generated much controversy in the UNIX world. On one hand, people argue that by using the *.rhosts* file, you are reducing or even alleviating the number of times a user's clear text password is transmitted across the network, which is considered a plus. On the other hand, if you use and allow *.rhosts* files, you may increase the possibility of your system being compromised by Internet intruders. At the Numerical Aerodynamic Simulation (NAS) Facility at NASA Ames Research Center, users depend on *.rhosts* files to run remote interactive programs and to remotely login without the use of passwords. At the same time, security is a paramount concern at NAS. This paper describes the methods to monitor and manage users' *.rhosts* files at the NAS facility.

Introduction

In today's large, multi-system environments it is not uncommon for users to have accounts on multiple systems at a single site or accounts on multiple systems at multiple sites. The proliferation of accounts causes a number of problems for users and system administrators. From the users' standpoint, having a large number of accounts can make it difficult to remember the password for each account. Some users get around this problem by using the same password for all of their accounts, which is a very poor security practice. Another problem from the users' standpoint is how to transfer files, when needed, between the different hosts. One solution to both of these problems is to have the different systems "trust" each other. This would alleviate the need to use or even remember the password for each account. On most standard UNIX systems today, there are two methods to accomplish this trust. One method is to use a configuration file called */etc/hosts.equiv*, which lists all of the hosts which are to be implicitly trusted by the local host. The other method is to use a user configurable file call *.rhosts*, which is located in the user's home directory. The *.rhosts* file acts as a personal */etc/hosts.equiv* file. Both of these files are consulted when using any one of the r-commands (*rlogin*, *rsh*, *rcp* and *rcmd*).

Issues of Providing Trust

The issue of "trust" between various systems presents a whole set of problems for the system administrator or security analyst. The use of these two files, especially the */etc/hosts.equiv* file, is considered a poor security practice by many people in the UNIX world. One vendor's man page for *.rhosts* even says "Use of the *.rhosts* file presents a security risk; use the file very cautiously or not at all in situations where security is a concern." David Curry, in his paper "Improving the Security of Your UNIX System", states "The only secure way to manage rhosts files is to completely disallow them on the system."

While the use of the */etc/hosts.equiv* file is almost universally seen as an evil by system administrators and security analysts alike, there are two schools of thought on the use of *.rhosts* files. On one hand, people argue that by using the *.rhosts* file, you are reducing or even alleviating the number of times a user's clear text password is transmitted across the network, which is considered a plus. Of course, if you are using Kerberos or something similar, the transmission of a clear text password over the network is not an issue. On the other hand, if you use and allow *.rhosts* files, you may increase the possibility of your system being compromised by Internet intruders. If an intruder is able to break into a host that a user on your system has

an *.rhosts* entry for, the intruder has an open door into your system if the path of vulnerability, via the *.rhosts* file, is discovered. This paper will present an in-depth analysis on the pros and cons of using *.rhosts* files, some typical types of *.rhosts* entries, and some methods to monitor and manage users' *.rhosts* files.

To start off the discussion, I ask the question, "Should you allow *.rhosts* files at your site?" The answer to this question depends largely on the specifics of the configuration at the site. Is the internal network gatewayed to the Internet via a firewall system? Is the internal network connected to the Internet? Are the internal users trusted? Is security a major factor at your site? Is Kerberos available or already installed at the site? Does the type of work performed by the users at your site require "trust" among the internal hosts or remote hosts on the Internet? These are some of the questions you need to ask yourself before you can determine if the use of *.rhosts* files is appropriate for your site. From the users' standpoint, the use of *.rhosts* files is very beneficial. The trust mechanism provided by the file allows them to remote copy (*rcp*) files from one host to another. It allows them to remote login without needing a password or having to remember a large number of passwords. The *.rhosts* file also allows users to run remote interactive programs. For the latter function, the use of *.rhosts* files is imperative.

At the NAS facility, it has been necessary to allow users the freedom to use *.rhosts* files, should they want or need the functionality. The NAS facility is comprised of over 300 computer systems running some flavor of the UNIX operating system. These systems are interconnected into a heterogeneous network using various types of network hardware and are supported by over 100 personnel who provide ongoing support and software development. The NAS facility, which primarily provides supercomputing services to other NASA sites, large aerospace corporations and a large number of universities, has over 1500 users nationwide. Most of the NAS users are at remote locations, and many of them run remote, interactive graphics or plotting packages from their remote sites. Hence, the *.rhosts* file is a crucial part of their daily work.

If you have made the decision to disallow *.rhosts* files at your site, then there are several precautions that can be taken to ensure users do not attempt to create one in their home directory. The simplest action to take would be to run a cron job (daily or weekly) which searches for *.rhosts* files and removes them. A more permanent solution would be to modify the sources for *rshd* and *rlogin*d so they ignore the *.rhosts* file and only consult the */etc/hosts.equiv* file. .

Types of *.rhosts* File Entries

Once you have made the determination that *.rhosts* files will be allowed at your site, you need to decide what types of *.rhosts* entries will be allowed. Before discussing the different types of good vs. bad entries, let's examine the format of the *.rhosts* file and the different types of entries which can be used.

The format of the *.rhosts* file is almost the same as that of the *hosts.equiv* file. Each line contains a hostname [username] pair or a special case entry. The system name can be the hostname of the remote system, the IP address of the remote system, or a netgroup name on the remote system. The hostname included in the entry must be the official hostname of the system and not some alias or secondary name. The username can be the name of any user on the remote host, the name of a netgroup, or the field can be left blank, in which case, the username defaults to the user of the account on the local host. The ability to classify a group of systems or users into a "netgroup" is a function of NIS (Sun's Network Information Services). The *.rhosts* file also allows the use of a special designator (a "+") in the hostname or username field. The "+" provides trust for all hosts or all users from the host specified. See *Figure 1* for an example of typical *.rhosts* file entries.

```
foobar.nas.nasa.gov crabb
badboy.nas.nasa.gov crabb
mustang.arc.nasa.gov crabb
```

Figure 1: Typical *.rhosts* file entries

The above entries would allow the user "crabb" to remotely access the local hosts from foobar, badboy and mustang.arc.nasa.gov without needing to supply a password.

Now that you have some familiarity with the types of entries found in a *.rhosts* file, how do you decide what should be allowed or disallowed? Again, this will depend largely on the configuration of your site and the desired level of security. Prior to the Morris Worm experiment, it was quite common for sites to provide trust between all hosts on the local network via the */etc/hosts.equiv* file. This was the case at the NAS facility. However, many sites no longer allow the use of the file to provide unilateral trusts on the local network. Generally, it is safe to allow users to add *.rhosts* entries for other systems on the local network; however, the entries should only allow logins for the local user. Account sharing via the use of *.rhosts* entries is very popular at UNIX sites. At NAS I frequently find users who are sharing their accounts via *.rhosts* entries.

The next level of *.rhosts* entries would include those for non-local systems (e.g., systems outside of the local domain). In a site that uses a firewall to connect to the outside world, it would not make sense to have such entries in the *.rhosts* file. The same would apply to sites using host-based filtering which filters out all non-local domain connections. However, at a site which allows non-restricted access to and from the Internet, the use of non-local *.rhosts* entries may be appropriate or even needed. At the NAS facility, many of our users are at remote sites, and as a part of their daily work, they may run remote, interactive programs which require trust between the local and remote system. As such, we allow *.rhosts* entries for non-local domain systems on our main systems. Access to NAS workstations is restricted to the local domain, hence non-local *.rhosts* entries on NAS workstations are non-functional. The next, and final level of *.rhosts* entries would include the use of the special designator "+". The "+" designator can be used in the hostname or username fields to add trust for all hosts or all users. This type of entry is the most insecure, and should never be allowed. The "+" designator is not allowed at the NAS facility.

When deciding if you want to allow *.rhosts* files, you should also consider whether *.rhosts* files will be allowed for the *root* account or any other system account. Many system administrators and security analysts may consider to use of *root .rhosts* files an extreme evil to be avoided at all costs. However, in a large heterogeneous environment, many system administration tasks would be very difficult or impossible without the use of *root .rhosts* files. At the NAS facility, *root .rhosts* files are used on the workstations, but are not allowed on mainframe systems such as the Convex and Crays. If you do allow the use of *root* account (or any uid 0 account) *.rhosts* files, you should minimize their use and only allow *root* entries for a limited number of systems in the local domain. For example, if you have a small number of file servers and a large number of workstations, you might want to use a *.rhosts* file that would provide trust for all file servers among the workstations and file servers.

Once you have determined what kinds of *.rhosts* entries will be allowed, you need to outline these in a formal policy, which is distributed to all users. The policy should also state the required permissions on the *.rhosts* file, what type of entries are explicitly not allowed, and what actions will be taken against users who violate the policy. At the NAS facility, we have a policy which covers the use of *.rhosts* files, */etc/hosts.equiv* files and batch configuration files such as the *.netrc* file. The files are monitored on a regular basis and appropriate action is taken against users who violate the policy.

Methods to Manage and Audit *.rhosts* Files

One of the major drawbacks of using or allowing *.rhosts* files is that they are often abused by users and targeted by intruders. Users frequently add entries for other users to allow them access to their account or files (e.g., when doing an *rcp*). Sometimes, out of ignorance, users will even add a "+" entry to their *.rhosts* file. Intruders who break into systems often use the *.rhosts* file as a means to provide themselves a back door into the systems for future "visits". The intruders will add a *.rhosts* file for a system account such a *bin* or *daemon* or they may add an entry into a regular user's account. Once a user creates a *.rhosts* file, the file is usually forgotten, and the contents are rarely looked at. Hence, should an intruder gain access to the system and add a *.rhosts* entry, it may go unnoticed for weeks or months. This actually happened at NAS on one

occasion involving a break-in from Australia. The intruders added a *.rhosts* file for the *bin* account with a single entry of the form "+", and they also added a "+" entry to several users' accounts. Fortunately, the incident and *the .rhosts* entries were discovered within several weeks.

Due to the increased risk that the use of *.rhosts* files bring, it is apparent that some method of monitoring and managing all users' *.rhosts* files is needed. The remainder of this paper will discuss an *.rhosts* file auditing program written and used at the NAS facility, and actions taken against users who abuse the policy. But first, I would like to provide a little background information and motivation for writing the audit program. Back in early 1990, I received a phone call from Dan Farmer (author of cops), who was working at CERT at the time. He was considering a joint project with a friend which involved doing an analysis of users' *.rhosts* files. He was interested in getting a copy of as many NAS users' *.rhosts* files as he could. Dan and his partner were interested in obtaining a large sample of files to see just how many users were using the *.rhosts* facility and what type of entries they were adding. One of the items Dan was interested in was determining how many *.rhosts* files were located on a single system -- sort of a "the security of this system is reliant on the security of N systems".

Since a user's *.rhosts* file was considered sensitive information, I told Dan that I would not provide him copies, but instead I would be glad to provide him the statistics. Out of curiosity, I wrote a quick shell program that would tell me how many users had *.rhosts* files and how many entries each user had and whether the entries were for systems that were included in the */etc/hosts.equiv* file or not. I ran my primitive script on few of the systems at NAS and was shocked at the numbers. Some users had 20-30 *.rhosts* entries for systems all over the Internet.

After this eye-opening experiment, I decided that a more extensive auditing program was needed. The initial *raudit* program was written in the Bourne shell. The first version of the program collected and reported on the following items for each user and for each host: the total number of *.rhosts* entries, the number of non-operational entries (i.e., hosts which were included in the */etc/hosts.equiv* file), the number of remote entries (i.e., hosts that were not in the local domain), the number of entries that were either malformed or possibly illegal (e.g. the username in the entry did not match the local user name). This program was run once a week on all NAS systems via a cron job. Due to the popularity of *.rhosts* files at NAS, the weekly output from the *raudit* run was close to 300 reports a week. Some reports were 5-10 pages long. See *figure 2* for an example of a full *raudit* run on a local workstation.

```
Rhosts File Audit Report for Chaos
==========================================
Summary Report For User: crabb
# of .rhosts entries: 2
# of non-operational entries: 0
# of non-NAS entries: 0
# of possible illegal entries: 0
==========================================
Summary Report For User: kensiski
# of .rhosts entries: 69
# of non-operational entries: 12
# of non-NAS entries: 8
# of possible illegal entries: 0
==========================================
Summary Report for Host: chaos
--------------------------------
The total number of rhosts files is: 2
The total number of .rhosts entries is: 71
The total number of non-operational entries is: 12
The total number of remote entries is: 8
The total number of possible illegal entries is: 0
```

Figure 2: Example of a long form *raudit* report

The first run of the report generated a large number of entries where the username in the *.rhosts* file did not match the user's name who owned the account. The process of determining if each of these entries was really a case of account sharing or just a case of different login ids for the same user, was a very arduous and time-consuming task. For each questionable entry, I would look up the user's full name in our user database to determine if the *.rhosts* username was an alternate login id for the same user. If the information from our user database did not provide a sufficient answer, I would do a *finger* of the *.rhosts* username at the remote host. In many cases this would provide an answer. If the *finger* daemon was not enabled on the remote host, I would *telnet* to the smtp port and do a *vrfy* on the *.rhosts* username. If these measures failed, I would send email to the user requesting verification that the *.rhosts* entry was valid. If the entry was valid, I would make note of it for future reference.

After several months of running with the initial version, it was apparent that much of the information being reported was not necessary or even useful in a weekly report. Also, the method I was using to keep track of all the alternate login names for each user was inefficient. What I was really needed was a list of possible illegal entries for each user. Hence, version two was inspired. The second version of *raudit*, written in Perl, incorporated the use of an alternate login id database, and the use of command line options. Version two has the option to print a short report of just illegal ("bad") entries. Since this version incorporates the use of an alternate login id database, the weekly reports show truly illegal entries, and in some cases, new accounts where alternate login ids need to be added to the alternates database. See *figure*3 for an example of an *raudit* short report (bad entries only):

```
Rhosts File Audit Report For foobar
-----------------------------------
WARNING: Possible illegal or malformed .rhosts entries for user johndoe:
 sunny.larc.nasa.gov majdi

WARNING: Possible illegal or malformed .rhosts entries for user janedoe:
 uxh.cso.uiuc.edu aae391ac

WARNING: Possible illegal or malformed .rhosts entries for user jsmith:
 grace.nas.nasa.gov root
 grace root

WARNING: Possible illegal or malformed .rhosts entries for user jnsmith:
 rtccd.arc.nasa.gov *
```

Figure 3: Example of an short form *ruadit* report

Version two also incorporates an option to search *.rhosts* files for a key word, such as a specific hostname. Prior to version two there had been several incidents involving systems in the *arc.nasa.gov* domain. When auditing the NAS systems to see if there had been any attempts to break in, one of the items I was interested in was to know if any NAS users had *.rhosts* entries for hosts which had been compromised on the *arc.nasa.gov* sites or some other domain. At the time I didn't have an easy method to accomplish this task on over 300 systems. However, the task lent itself well to a feature of the *raudit* program. The command line to run *raudit* to find a match for a hostname is: *raudit -m -h host_keyword.* The host keyword can be the short hostname, the fully qualified name or any sub-string. Sometimes I use this feature to locate *.rhosts* entries for a specific host. This feature of the *raudit* program has been very useful in the follow-up of security incidents. With just a few commands, I can locate all host entries for a specific host or domain from all systems at the NAS facility.

Once I populated the alternate login id database, reading and processing of the weekly *raudit* reports became a much simpler task. The master copy of the alternates database and *raudit* program are kept in the master source tree on a file server. The alternates database is updated as the reports are read and processed each week. Each week the alternates database and the *raudit* program are distributed to all systems before

the cron job to run *raudit* executes.

Auditing and managing of *root .rhosts* entries is another issue. At the NAS facility we allow *root .rhosts* entries on all workstations and the file servers. Instead of auditing the *root* account *.rhosts* files on a weekly or daily basis, we have a nightly cron job which reinstalls the *root .rhost* files with a master copy. All workstation *root .rhosts* files are the same. This method of auditing was chosen to reduce the need for human interaction. Also, replacing the *root .rhosts* file on a nightly basis is helpful because sometimes during the course of system work or testing, the *root .rhosts* file will be modified to add a temporary entry. Occasionally, the support people will forget to remove these "temporary" entries. Replacing the file on a nightly basis ensures temporary entries are really temporary. Non-root entries in *root .rhosts* files are not allowed at the NAS facility.

Methods To Handle Abusers Of The Policy

Invariably, users will abuse the *.rhosts* file policy, either intentionally or out of ignorance. Appropriate action must be taken against the policy abusers to ensure that future infractions do not occur. A policy regarding the use of *.rhosts* files and what are the allowable entries should be clearly defined and provided to all users on the system. The policy should also state what action will be taken against users who abuse the policy. A written policy will also provide justification for the actions taken against people who abuse the policy.

At the NAS facility, the action taken against policy abusers depends on the severity of the violation. The NAS account policy states that account sharing is not allowed and that adding *.rhosts* entries for other users is considered account sharing. Per policy, we have the right to disable a user's account for suspected account sharing. However, when the account sharing is via a *.rhosts* file entry, the user will get one or two warnings prior to the account being disabled. If the entry in question provides trust for what appears to be a general use account (e.g., *guest* or *visitor*) at a remote site, I remove the entry immediately, and send the user an message such as the one shown in *figure* 4.

```
Mr. Doe -
The following illegal .rhosts entry was REMOVED from your .rhosts on wk00:

Rhosts File Audit Report For wk00
-----------------------------------
WARNING: Possible illegal or malformed .rhosts entries for user johndoe:
36.65.0.120 guest # ctf unix

Adding entries for other users, especially guest accounts, is against
NAS policy. Please do not add such entries in the future or your NAS
accounts will be disabled.

Michele Crabb, crabb@nas.nasa.gov
Computer Security Analyst/Distributed System Support
Sterling Software Incorporated/ NASA-Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035
(415) 604-4337
```

Figure 4:Example email message sent to users with *.rhosts* entries for general use accounts

The user is given one email warning about adding such entries in the future. If the user adds the same entry again or another illegal entry, the user's account will be disabled. Per NAS policy, we can disable the user's account for up to three days before attempting to make contact. Depending on what the user has to say when we call, the account will be re-enabled or remain disabled.

If the entry in question provides trust for another user at the NAS facility, I will send email to both users. Usually, in these types of cases, the users add entries for each other in their *.rhosts* files. The users will be warned via email that future infractions will result in both accounts being disabled. The users are given one week to remove the entry. If the entry is present at the next *raudit* report, then I will remove the entry and attempt to contact the users by phone. In some cases, the users have not read their email. If the *.rhosts* entry appears to be for another user at a remote site, I will send an email warning message to the local NAS user requesting the entry be removed. The user is given one week to remove the entry. As in the case above, if the entry is reported on the next *raudit* report, I will manually remove the entry, and try to contact the user by phone. A typical email message sent to users shown in *figure* 5.

```
Hi -
You have the following illegal .rhosts entries on wilbur:

WARNING: Possible illegal or malformed .rhosts entries for user janedoe:
catfish.cs.umd.edu robertz
hyena.cs.umd.edu bkmone

Adding .rhosts entries for users other than yourself is against NAS policy.
Please remove these entries and any others like them on any of your NAS
accounts as soon as possible. Adding such entries in the future may result
in having your NAS accounts disabled.
```

Figure 5: Email message sent to users with illegal *.rhosts* entries

All email correspondences sent to users regarding *.rhosts* entry violations are archived. New reports of illegal entries are checked against the archived mail. If a user re-adds an entry that was previously declared illegal (or a similar entry) weeks or months after the first warning, the user's account will be disabled for a minimum period of three days. Users can permanently lose their NAS account privileges over *.rhosts* policy violations, although there have yet to be any cases.

When corresponding to users who violate the *.rhosts* file policy, I suggest other methods the user can use to accomplish the needed tasks. In some cases, it may require more work on the user's part, but the user understands the policy must be followed. Many times I receive email replies back from the users apologizing for the entries which include various explanations. Frequently, the users will tell me the entries were added by mistake. Some users have told me that the other user never logs into the account, but only uses the *.rhosts* entry to remote copy files back and forth. There have been a number of a cases where users at universities add *.rhosts* entries for their "assistants". In these cases, I inform the users (usually a project Principal Investigator) they need to request accounts for their assistants as well. I even had one case where the owner of the account was rarely using his account. Instead, his two "assistants", from two different universities, were using the account. The owner of the account didn't think he was violating any policy. Some of the stories I have heard from users regarding their illegal *.rhosts* entries have been quite amusing. In some sense, I know how a police officer feels when listening to the different excuses people provide for speeding.

Future Directions

Ideally, it would be nice to have an alternate means of providing login authentication without the use *.rhosts* files or the sending clear text passwords over the network. One solution to this problem is the use of the Kerberos software or something similar. Another solution would be to use the challenge and response cards, such as the SecureID card. The NAS facility is currently considering these two possibilities. However, for very large sites it may be difficult to install Kerberos on the growing number of systems and platforms, and it might be difficult to provide challenge and response cards to a user base that exceeds a thousand. Until there are more cost effective solutions, the use of the *.rhosts* file will remain an issue system administrators and security analysts will have to address.

There are several modifications that could be made to the *raudit* program to improve functionality and performance. The current method used to store and search the alternates login id database is inefficient. When I first developed the idea of the alternates database, I was unaware that it would grow to its current size. This feature of the *raudit* program needs to be redesigned. As a part of the maintenance of the alternates database, it would be nice to have some function which could attempt to do a verification of *.rhosts* entries where the user has different login ids on different hosts. For example, a function which could do a *telnet* to the smtp port to do a *vrfy* command on the login id, would be a solution. Another improvement related to the use of the alternates database is the need for an automated method to know when a user in the database needs to be deleted. Currently, I have to manually check the alternates database against the official NAS user database to ensure archived users are removed. Although, this does not cause any problems with the *ruadit* program, reducing the size of the database would reduce the execution time of the program. A method to validate a hostname would also be a helpful feature. I have found cases where users have entries in their *.rhosts* files for hosts which no longer exist.

Another idea I have been considering for the management of *.rhosts* files and the *ruadit* program is to limit the number of non-local *.rhosts* entries a user would be allowed to have. There would be no limit on local *.rhosts* entries; however, some small finite number of remote entries, such as five or maybe ten would make the process of auditing and managing *.rhosts* files much easier. If such a scheme were to be implemented, users could specify which remote hosts they wish to use in their *.rhosts* files on their account request forms. Users could request additions, modifications, and deletions to their list of allowed hosts. This information could be stored in a database that would be accessed by the *raudit* program. If a user were to add an entry that was not previously requested, it would be reported as a illegal entry by the *raudit* program. Minimizing the number of *.rhosts* entries for remote hosts would also reduce risk level to the local network.

Availability

The *raudit* program is freely available from the NAS facility. To receive a copy via email, send an email request to *doc-center@nas.nasa.gov*. If you have any questions regarding the use of the *raudit* program or how security is handled at the NAS facility, send email to *crabb@nas.nasa.gov*.

Author Information

Michele Crabb has been the primary computer security analyst for the NAS Facility at NASA Ames Research Center for over four years. During her nine years at Ames, Michele has worked in several divisions, in a variety of positions ranging from applications programming to UNIX system support. Prior to becoming the NAS security analyst, she was actively involved in providing system administration support for the large number of workstations at the NAS facility. Michele can be reached via electronic mail at *crabb@nas.nasa.gov*, or via US Mail at NASA Ames Research Center, Mail Stop 258-6, Moffett Field, CA. 94035-1000.

Campus Email for Everyone: Making It Work in Real Life

Stephen Campbell

Dartmouth College
Hanover, NH 03755-3523
Stephen.Campbell@Dartmouth.EDU

Abstract

How do you build an electronic mail system for an eclectic mix of people with vastly differing needs for communications, win them over, and keep them coming back for more? From the early days of timesharing to today's mix of Macintosh® computers, workstations, and mainframes, we at Dartmouth College have developed an email system that serves everyone on campus — even people who don't use computers. This paper describes that email system and discusses some of the non-technical issues that it raises. The campus network is a blend of LocalTalk® and ethernet. The central component of the email system is a Macintosh-based client/server mail system called BlitzMail®. The Dartmouth Name Directory, a name service that includes all students, faculty, and staff, is essential to BlitzMail and to implementing campus wide addressing-by-name. A system for printing email messages and delivering them through the campus mail system brings email to those who do not use a computer. Beyond the hardware and software are questions of support, email citizenship, and the impact of a pervasive email system on the institution.

The Dartmouth Environment

Dartmouth comprises a private four-year liberal arts college and professional schools of business, engineering, and medicine, plus a strong association with the Dartmouth-Hitchcock Medical Center. In 1993 there were 4,300 undergraduates and 1,200 graduate and professional students, 900 faculty and 2,200 staff employees of the college and professional schools, and 4,300 employees of the medical center. About 85 percent of this population — over 10,000 people — use the Dartmouth email system.

At present and for the past decade the Macintosh has been the personal computer of choice at Dartmouth. Almost all students purchase one when they arrive, and Macs are found in nearly all faculty and administrative offices. Unix® — and to a lesser degree VMS® — workstations appear in some departments and in the engineering and medical schools. The business school and the medical areas use DOS machines and Unix workstations. The College's Computing Services organization operates Unix, VMS, VM/CMS, and Dartmouth College Timesharing System (DCTS) host systems providing academic, administrative and general-purpose computing for the college. Computing Services also manages the campus network and provides central Macintosh email servers for the college and the medical center. The medical center operates its own network and central computing.

How We Got Here

The first email system at Dartmouth was a traditional line-oriented mail program on the Dartmouth College Timesharing System. Its Teletypes terminals and serial network were used mostly by students, faculty, and staff of the Computer Science department and Computing Services. The computer had no off-campus links.

In 1983 the first Unix system arrived: a VAX™ 11/750 running BSD Unix. With it came the first link to the rest of the world: a dialup UUCP connection. Other systems followed — VMS and CMS — but email

remained line-oriented and networking consisted of cobbled-up RS232 host connections quaintly dubbed "milking machines."

Late in 1983 Dartmouth evaluated the then new Macintosh computer and early the next year selected it as the recommended personal computer for students, faculty, and staff. The Mac had a superior user interface, came with the most sophisticated networking then available for personal computers, and cost about $1,200 for a reasonable entry-level machine. The college committed to converting the campus network to AppleTalk® and wiring all dormitory rooms. This effort began in the late spring of 1984 and was complete by the beginning of the fall term. Installing the 2,600 LocalTalk ports cost $750,000 for wire, network routers, and labor, representing a per-port cost of about $290.

In the fall of 1984 the students could use their Macs as word processors and could use DarTerminal, a Dartmouth-developed terminal emulator, to log onto the central computers. But most students had accounts only on DCTS, and that system's original mail program remained the only email system available on campus.

By 1987 the limitations of this mail system were evident. Dartmouth had evaluated commercial email packages for the Macintosh but found them unable to scale up to thousands of users. On November 1 a group of college software developers started work on a client/server Macintosh mail system, and after a two-month blitz of effort, they had a prototype system with a Mac client with a server and name directory hosted on the DCTS mainframe. They called the package "blitz mail," and the name stuck.

In the late 1980s an ethernet network that began with the Unix systems has grown to cover much of the campus and connect Dartmouth to the Internet, and the development of an AppleTalk/TCP gateway has blended the two networks. In 1989 we ported the BlitzMail servers to more flexible and inexpensive Unix platforms, making it possible, for example, to access the name directory from workstations and mainframes. The college has extended campus wiring to include most administrative buildings and many fraternities and sororities. BlitzMail has continued to evolve, and today to many people BlitzMail *is* computing at Dartmouth.

Today's Dartmouth Network

The Dartmouth network today is a blend of LocalTalk and ethernet, of AppleTalk and TCP/IP, DECNET®, and IPX. Figure 1 is a simplified schematic of the college network. The network at the nearby Dartmouth-Hitchcock Medical Center (DHMC) is a similar blend. Out of habit we refer to the IP network and the AppleTalk network, but DECNET and IPX appear on the IP network, IP appears in parts of the AppleTalk network, and AppleTalk packets appear everywhere. The entire college network is managed by Computing Services; the DHMC Network Operations group manages the DHMC network.

The AppleTalk (AT) network, including the DHMC, contains about 150 AT zones and about 450 AT networks. On a busy day about 5,000 Macs are active on the network. AppleShare servers and LaserWriters® are managed by their owners, so we cannot know their numbers exactly. We estimate about 200 servers and 500 printers.

The AppleTalk routers are NEDCo's, processors designed and build by New England Digital Corporation to be used as the CPU in NED's music synthesizers. There are 100 of these unlikely but effective processors in the network. They cost about $6,000 each, hold a maximum of 128K bytes of memory, and are programmed in a PL/I-like language called DXPL. Dartmouth has used NEDCo's since 1979, but they are no longer manufactured and we are down to ten spares. Other than the fact that they are paid for, their primary advantage is their flexibility: we have designed RS232, X.25, and ethernet interface boards for them, and *we* program them.

The college IP network is a subnetted class-B network with about 15 active subnets and 700 hosts. The physical layer is a mixture of fiber, thicknet, thinnet, and 10baseT. We use cisco routers and our own AT/IP gateways for routing. We have programmed the NEDCo routers to tunnel IP packets through the AT network. Since the AT network extends to more campus buildings than the IP network, this gives us the ability to have remote IP subnets.

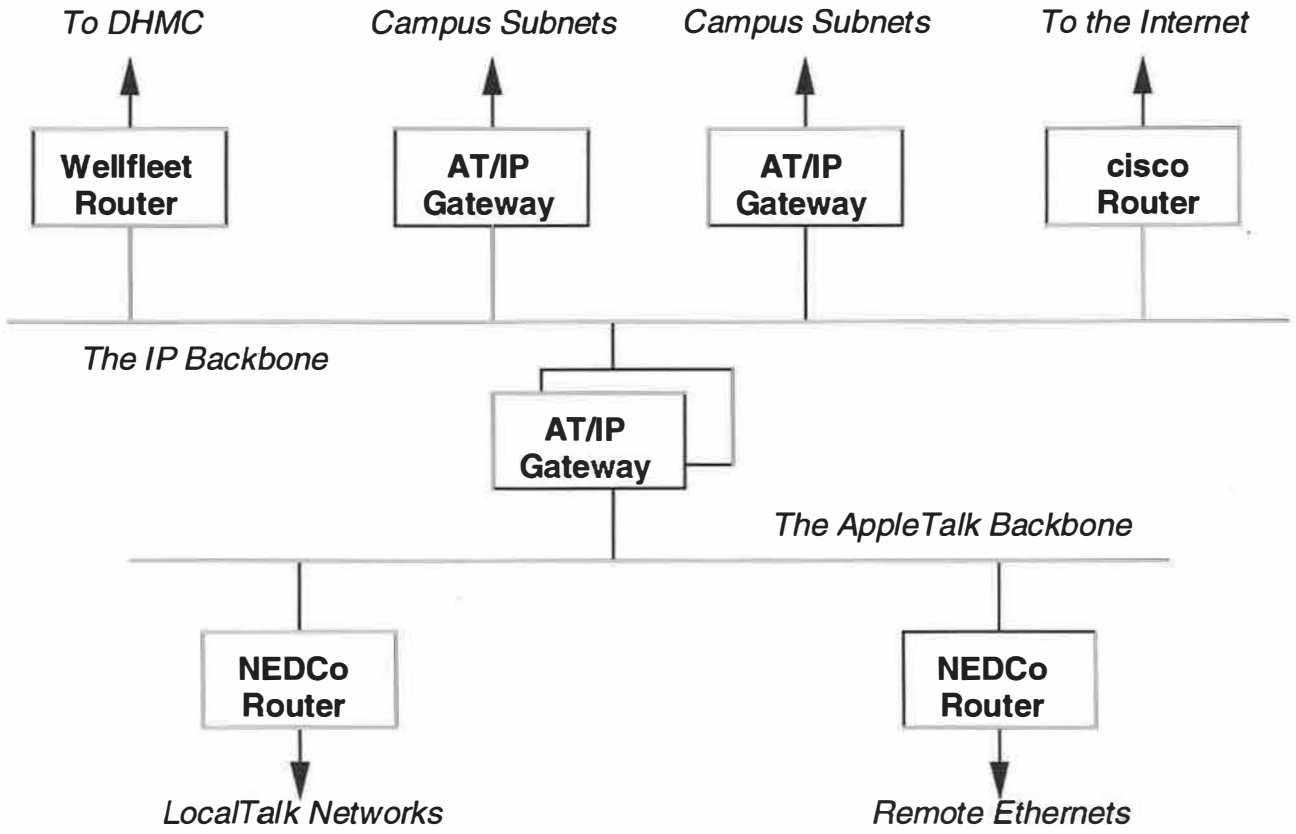


Figure 1
The Dartmouth College Network

The key component in the blending of the AT and IP networks is the AT/IP gateway. Dartmouth developed this device on a Mac II platform in 1989. It interconnects LocalTalk and ethernet. It is an AppleTalk extended router and an IP interior gateway. It performs interdomain name and address mapping and it translates TCP to and from the Dartmouth-developed AppleTalk stream protocol. Finally it supports KIP encapsulation of IP packets over LocalTalk, using dynamic or static IP address assignment. The AT/IP gateways make it possible for the BlitzMail Macintosh email client using only AppleTalk protocols to use a server on a Unix host that speaks only TCP/IP. In fact these gateways make it possible for us to refer simply to "the network" in most discussions. For example Figure 2 shows the simple topology of the email system. For email purposes the Macintoshes talk to the BlitzMail servers, and the servers, the mail hub, and the other hosts are TCP/IP and Internet peers.

The Dartmouth Name Directory

The Dartmouth Name Directory (DND) is the linchpin of the Dartmouth email system. The DND is a 17,500-entry database of every student, faculty member, and employee of the college and the medical center. There are also entries for most campus departments and organizations. The DND is used by the BlitzMail email system, by host mail systems, and by various non-mail applications.

The lookup algorithm is designed to make it possible to find a person's entry given any reasonable form of their name. A query can be any combination of tokens from the *name*, *nickname*, or *deptclass* field. The order of tokens is unimportant, the search is case-insensitive, and punctuation marks are ignored. Tokens can be omitted or written using only an initial substring, but at least one complete token must appear in the query. For example "Mary Jane Doe," "Mary Doe," or "M. Doe '94" are valid queries. There may be zero, one, or many DND entries matching a given query. Some applications such as email addressing require a single unique match, which, if necessary, can be achieved by adding a unique nickname to the entry.

Table 1 shows the important fields in a DND entry. In addition to the name and description of the fields, it shows who is allowed to read (retrieve) and write (change) each field. Each person's entry has a password which only that person (or the database administrator) may change. The user must provide the password in order to change the value of a field. Some fields, such as the user's class affiliation, can be changed only by the database administrator.

Field Name	Read	Write	Description
name	anyone	-	User's full name (indexed)
nickname	anyone	user	User's nicknames (indexed)
uid	anyone	-	College ID number (indexed)
deptclass	anyone	admin	Department or Class (indexed)
hinmanaddr	anyone	user	Campus mailbox
mailaddr	anyone	user	Preferred email address
phone	anyone	user	User's telephone number
blitzserv	anyone	admin	Name & zone of user's BlitzMail server
bullserv	anyone	admin	Name & zone of user's bulletin server
pw	-	user	Entry password

Table 1. DND Fields

A single server computer is home to the Dartmouth Name Directory. The first server was the DCTS system; the current version uses a NeXT® computer. The server uses a database manager to retrieve

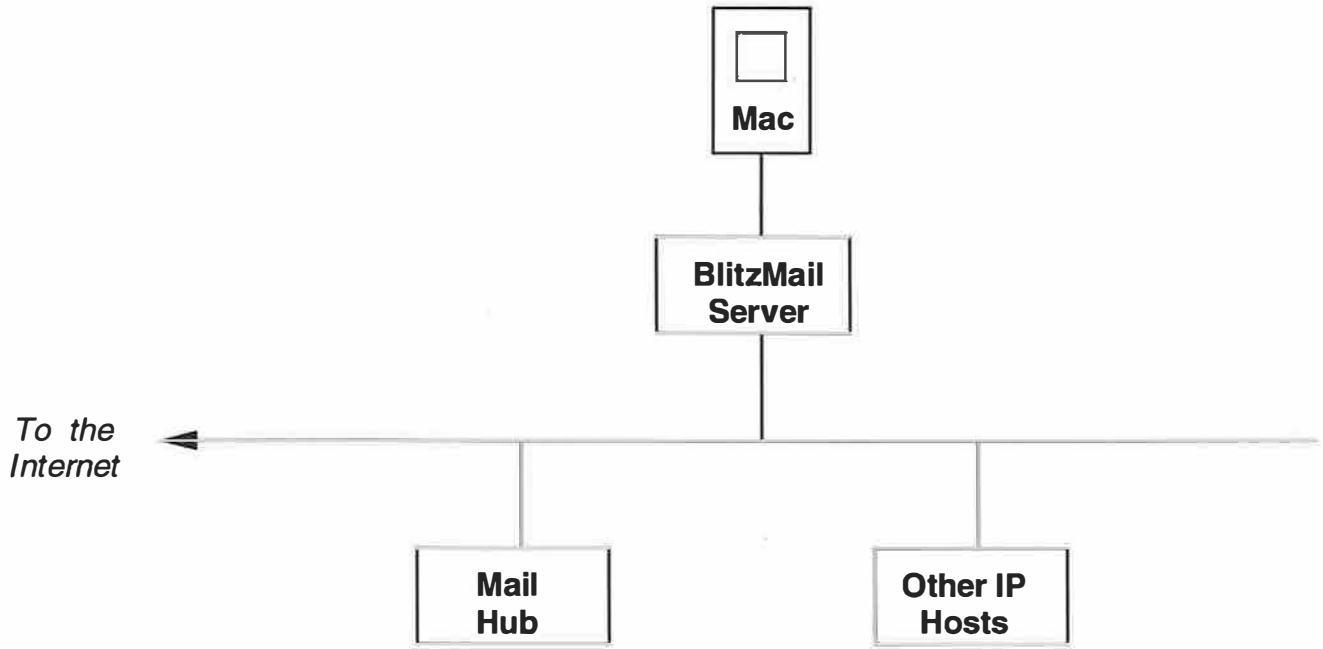


Figure 2
The Dartmouth Email Topology

potential matches, then applies additional tests to generate the final set of matches. The original server used a flat data file, which would still be reasonable if the database contained a few hundred entries. Later versions used Unix *dbm* files; the current version uses Oracle™ as the underlying database manager.

The DND is available to network applications via a stream protocol on a well-known TCP port on the server. The protocol is a series of ASCII commands and responses, similar to SMTP. Important commands in the protocol include LOOKUP to make queries, CHANGE to change field values, VALIDATE to validate a password, and CHPW to allow users to change their passwords. When a password is required for an operation, it is never sent across the network. Instead the server chooses a random string and sends it to the client, which DES-encrypts it with the password and returns the result to the server. The server compares this result to its own encryption of the string. If the two results match, the client is validated. A library of C functions is available to make it easy for applications use to the DND.

The DND administrator can make bulk additions and deletions to the database. These changes come from the registrar and the college and DHMC human resources offices. The administrator also makes individual corrections, additions, and deletions with a separate administrative interface. Users can change their own phone numbers, nicknames, and email addresses.

The original purpose of the DND was to validate BlitzMail users and to hold technical information needed by the BlitzMail server. New applications resulted in new fields being added to the database. These applications include extensions to BlitzMail and mail addressing-by-name for other campus email systems (both described below) as well as a *fingerd* server for the entire institution and a phone book server for Gopher. The DND has also become the de facto authority on who is "a Dartmouth person," and we use it as a general validation server and for software license enforcement.

Because the DND is critical to so many applications and having an entry in the DND grants access to these applications, it is important to have clear policies about who can be in the DND and who makes the decision. Entries for registered students and full-time faculty and staff are automatic. It becomes less clear for groups such as part-time employees, alumni, spouses and children, and contractors doing work for the college. Our policy is that we will create an entry for these people provided the entry is needed for college-related business and provided there is a directly affiliated person who will act as sponsor. Another issue is when to remove the entry of a student who graduates or an employee who leaves. We retain student entries for one term after graduation; this gives them time to make other arrangements for receiving email. For employees and faculty who leave, we have found that by the time the change information reaches us, the person has generally made other arrangements and it is safe to remove their entry.

BlitzMail

Seventy-five percent of the people at Dartmouth are email users. Ninety-five percent of those — about 13,000 people — use the Macintosh email system called BlitzMail. Figure 3 shows the BlitzMail system data flow. BlitzMail is a client/server system. The client program and the optional notification control panel and driver run on the user's Macintosh. The three server programs — the BlitzMail server, the DND server, and the notification server — run on Unix machines. Communications between the Mac-based programs and the servers is by AppleTalk stream protocol or by MacTCP. Communications among the servers is by TCP.

The user's mail messages, mailing lists, and some personal information such as reply-to address and folder sort orders are kept on the BlitzMail server. Only machine-specific user information such as preferred fonts reside on the user's local disk. This means a person can use BlitzMail from any networked Mac, not just their own.

Here is the sequence of events in a typical session. When the user starts up the BlitzMail client, it connects to the DND server at a well-known network address and retrieves the host name of the user's BlitzMail server. The client then connects to the designated BlitzMail server, the server validates the user by consulting the DND server. The client then checks for any messages in the user's "in box." The user can then read, compose, file, or delete messages. The BlitzMail server may query the DND server to resolve recipient addresses on newly composed messages. The user may edit her mailing lists or use the client's

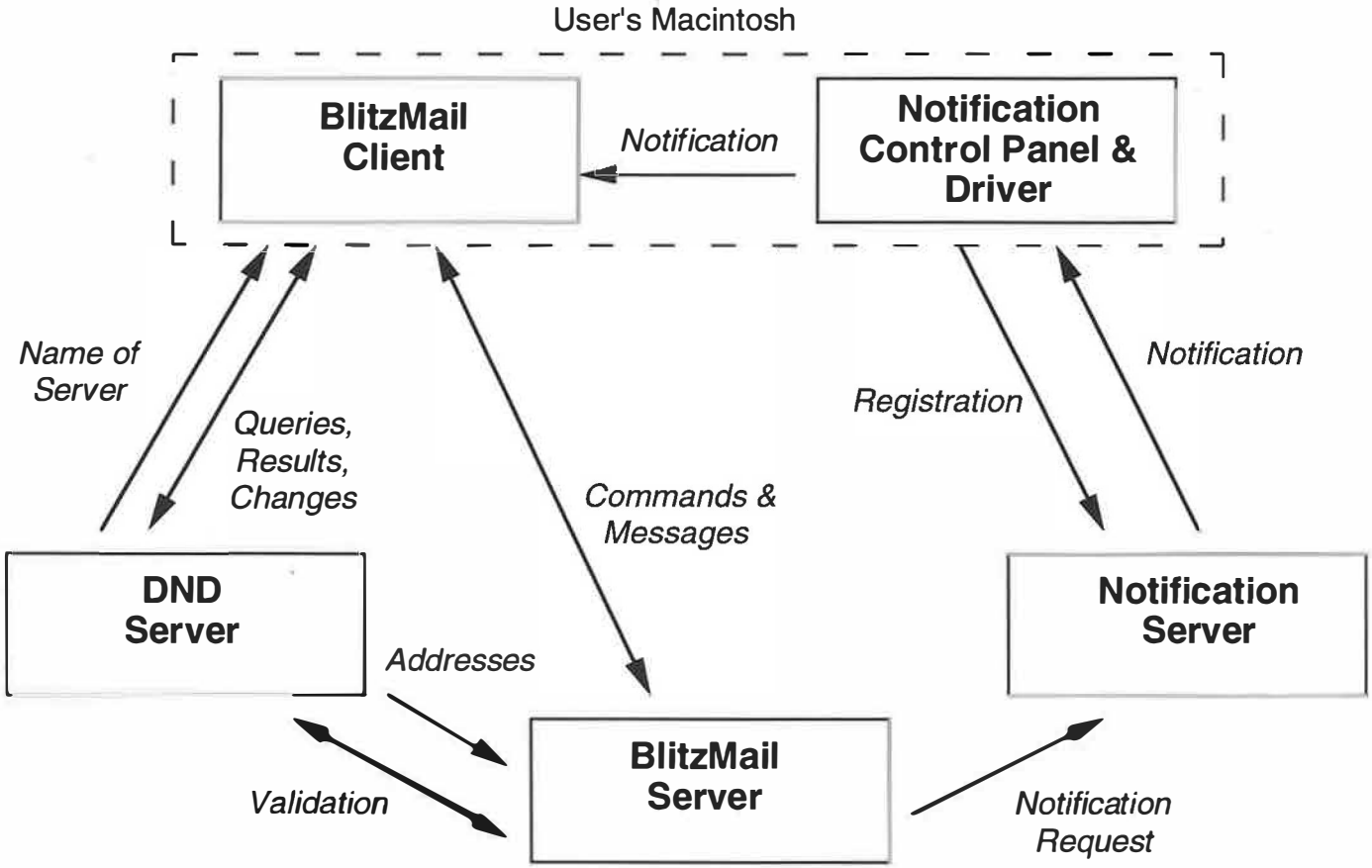


Figure 3
BlitzMail Data Flow

DND query menu to look people up or change her own DND information. At any time, whether or not the user is running the client, a new message may arrive on the server. The server then sends an arrival notice to the notification server, and if the notification server has a current registration for the user, it sends a notification to the notification driver in the user's Mac. The driver either notifies the client program if it is running or else beeps and flashes an icon that indicates new mail has arrived.

The BlitzMail Client

Like any mail user agent, the BlitzMail client program allows the user to read, reply to, compose, and file mail messages. Figure 4 shows a typical screen of the client showing the In Box window, a message window for a message from the in box, and a compose window for composing a reply to the message.

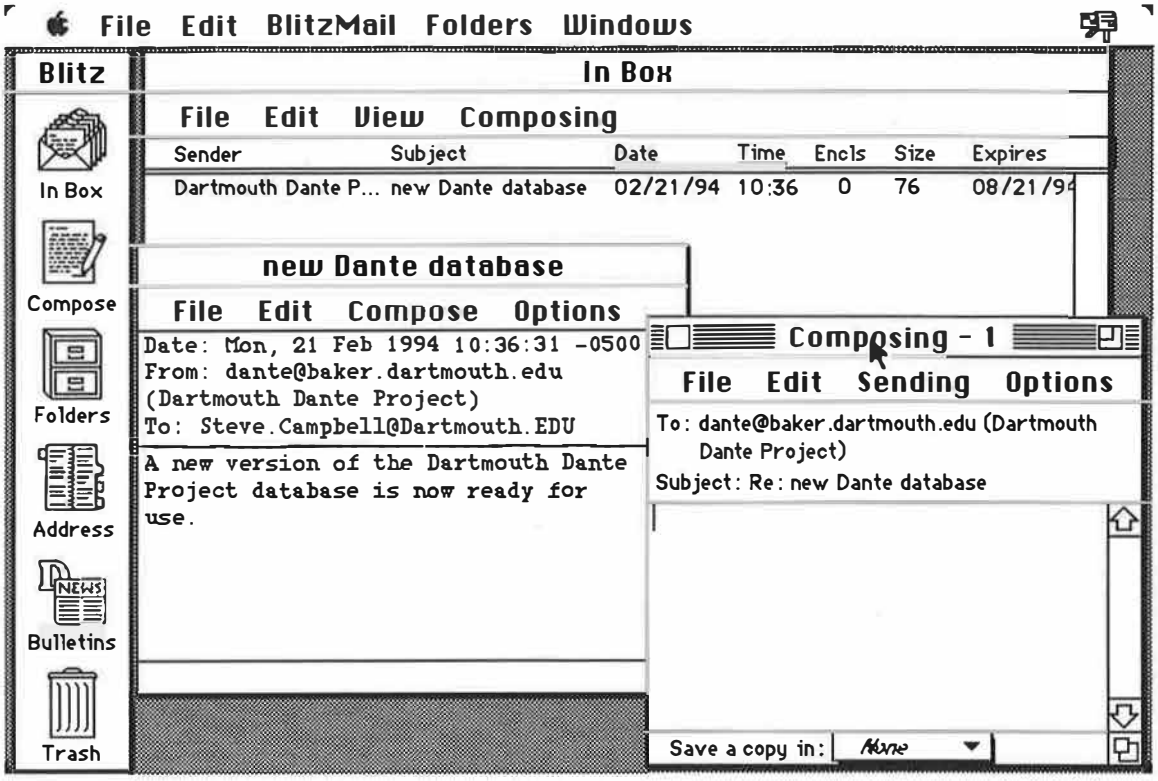


Figure 4. BlitzMail Client Screen

A summary of incoming messages appears in the In Box window, which the user can choose to have appear automatically if there are new messages, or which she can open with the In Box icon. The messages themselves are not yet retrieved from the server. In addition to sender, subject, date, and time (all derived from the message headers), the in box shows a bullet for unread messages, the size of the message (important if you are using a slow modem), the number of enclosures in the message, and the date on which the message will be purged from the server. Double-clicking on a particular message causes the client to retrieve the message (but not the enclosures) and display it in another window. From that window the users can save, reply-to, or delete the message, save any enclosures, or just close the window to keep the message in the in box. Replying to a message or clicking on the Compose icon opens a composing window. The user can type or cut and paste text, include recipient addresses from the address book, and add an arbitrary number of enclosures. A completed message can be sent, filed, or both. The user can define message folders, which reside on the server, for saving messages. Three special folders are automatically created and maintained for the user: the in box, recently sent messages, and recently deleted messages.

The enclosure mechanism allows any Macintosh file to be included in a BlitzMail message. The user adds the enclosures when the message is composed. They can be any Macintosh file, for example a program or a

word processor document. When the user tells the client program to send a message, the client transfers the enclosed files to the server along with the body of the message. When an incoming message contains enclosures, the user can tell the client program to save the enclosure. The client then reads the enclosures from the server and saves them on the Mac's local disk. Because *binhex* has become a standard for enclosing Macintosh documents, when a BlitzMail message is sent to a non-BlitzMail recipient, the server binhexes the enclosures and includes them in the message text. When the server receives a message for a BlitzMail user and the message contains binhexed text, the server unbinhexes it and creates an enclosure in the message.

BlitzMail offers several ways of specifying recipient addresses. If the recipient is a person or organization that is listed in the Dartmouth Name Directory, then their address is their name. The BlitzMail server queries the DND server for the person's "real" email address, which might be the recipient's BlitzMail server if the recipient is a BlitzMail user, or it might be a standard RFC822 domain address if the recipient uses some other email system. In any case, the person composing the message does not need to know where another Dartmouth person receives email. The composer may also type an RFC822 domain address directly; this would be required for recipients outside Dartmouth and is optional (and discouraged) for people on campus. BlitzMail also offers several types of email aliases and mailing lists. These can be personal aliases and lists known only to the individual user and kept in the user's mailbox on the server, or they may be public mailing lists that anyone can send to. Public lists may be open or closed, i.e. it may or may not be possible for users to see who is on a particular list.

The BlitzMail client offers the Macintosh user a convenient way to look up someone in the Dartmouth Name Directory or to edit their own DND information. The user can type a name in the lookup window and the server displays up to about a dozen matching names with their affiliation, phone number, and email address. In the editing window the user can change any of the DND fields that they are allowed to change.

The BlitzMail Server

The BlitzMail server program provides the message storage and forwarding functions needed by the BlitzMail client. The server receives incoming messages from other users on that server, from other BlitzMail servers, and from non-BlitzMail SMTP hosts. Messages and enclosures in any of the user's folders, including the in box, are kept in files on the server's disks and sent to the client on demand. When the user composes a message, the server receives it from the client and delivers or forwards it as needed. The server is also the repository for the user's personal email-related information such as mailing lists and optional forwarding instructions. The server uses this information in the course of handling messages and allows the user to display or change it on demand. The server is also responsible for transferring enclosures, including processing them for export and import.

The protocol for client/server communications is a line-oriented protocol similar to SMTP but with extensions for BlitzMail functions including transmission of binary information. It has been optimized to make reading and sending mail easy for the server and to make the process fast on low-speed dialup connections. The protocol for server-to-server communications is SMTP with extensions for binary transmission and for moving a user from one server to another. When the server needs to forward a message to a non-BlitzMail host, it opens a conventional SMTP session with a single designated forwarding host. At Dartmouth that host is the campus mail hub. By forwarding all out-bound mail to a single host, the server avoids complicated address parsing and MX handling. The server also listens on the standard SMTP port for incoming messages from other BlitzMail servers or from non-BlitzMail hosts.

There can be one or more BlitzMail servers in a system. Users are assigned to a server arbitrarily; a person's DND entry indicates which server they use. Dartmouth currently uses five NeXT machines and one Digital Equipment Corporation Alpha AXP® running OSF/1®. The server program, which can handle many simultaneous sessions, runs as one multi-threaded process, and these machines were chosen in part because their operating systems support multi-threading. We use a rule of thumb of one megabyte on the server for each client. The NeXT servers have about two gigabytes of disk storage each while the Alpha has about six gigabytes.

Dartmouth Bulletins

Several years after BlitzMail was introduced, users began asking for a mechanism that would provide them with time-critical messages about specific topics. The solution was Dartmouth Bulletins. Bulletins contain announcements from over 200 student, academic, and administrative organizations ranging from the Afro-American Society to the computer center to fraternities to the performing arts center to the Stargazers Club to the campus police.

Bulletins combine characteristics of mailing lists and USENET news readers. They are a one-to-many message system. Each bulletin topic has one or more moderators who may post to the topic. New moderators take a short course on how to use the bulletin system effectively, and new topics are reviewed before being created. Anyone may read bulletins using the same BlitzMail client with which they receive their email. Figure 5 shows a typical screen. When the user clicks the Bulletin icon, a list of all bulletin topics appears. By clicking on any topic, the user can read current bulletins on that topic. For more immediacy the user can choose to "monitor" the topic. When a moderator posts a bulletin to a monitored topic, the user receives an immediate notification of the new bulletin. Notification can be in the form of a beep and flashing icon, or a window for the topic may open automatically showing the new bulletin. Reading a bulletin is like reading an email message, and the usual functions such as saving, forwarding, and replying to the sender are available.

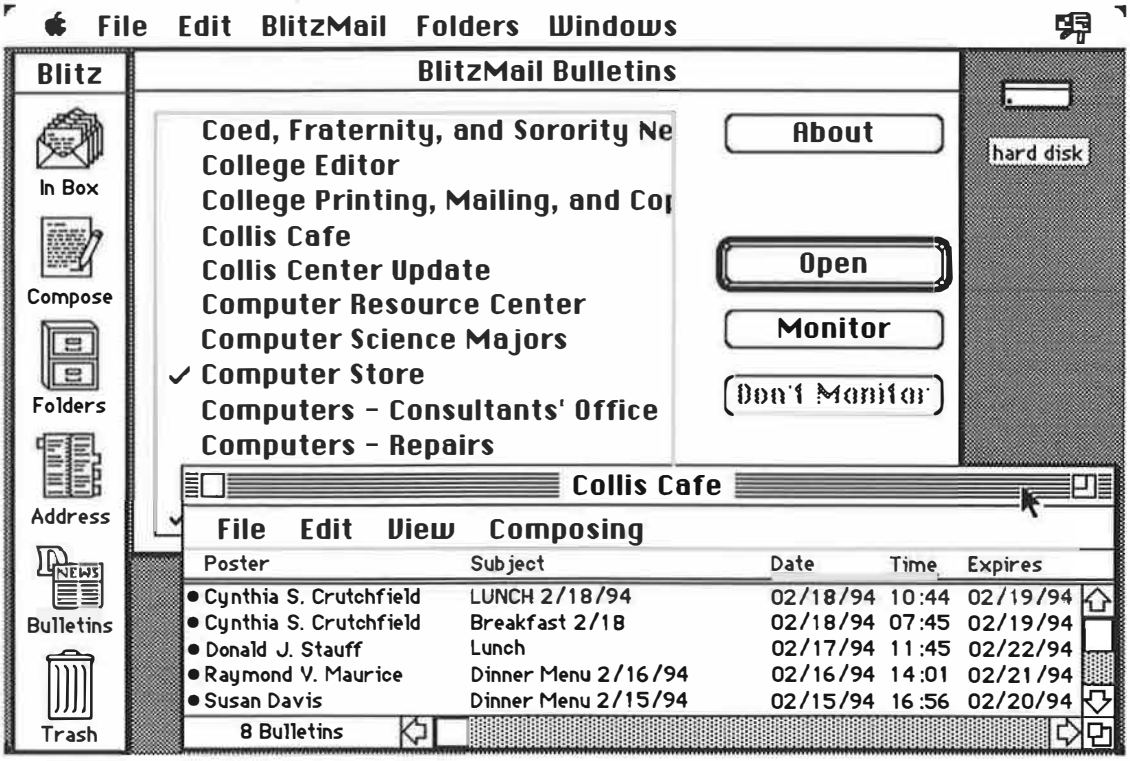


Figure 5. Dartmouth Bulletins Screen

Given Dartmouth Bulletins' similarity to USENET news, it was natural to use the news software to implement the bulletin server. Figure 6 shows the bulletin system data flow. Bulletin topics exist as conventional USENET moderated newsgroups on our campus news server. As such they can be read by anyone with any of the standard news reading programs. A special bulletin server program acts as an intermediary between the Macintosh BlitzMail client and the news system. The bulletin server maintains information about each BlitzMail user's bulletin reading history, i.e. which topics the user monitors and which bulletins he has already read. The server also supports a subset of the NNTP commands. When the user starts the BlitzMail client, the client contacts the server and asks if there are any new unread bulletins

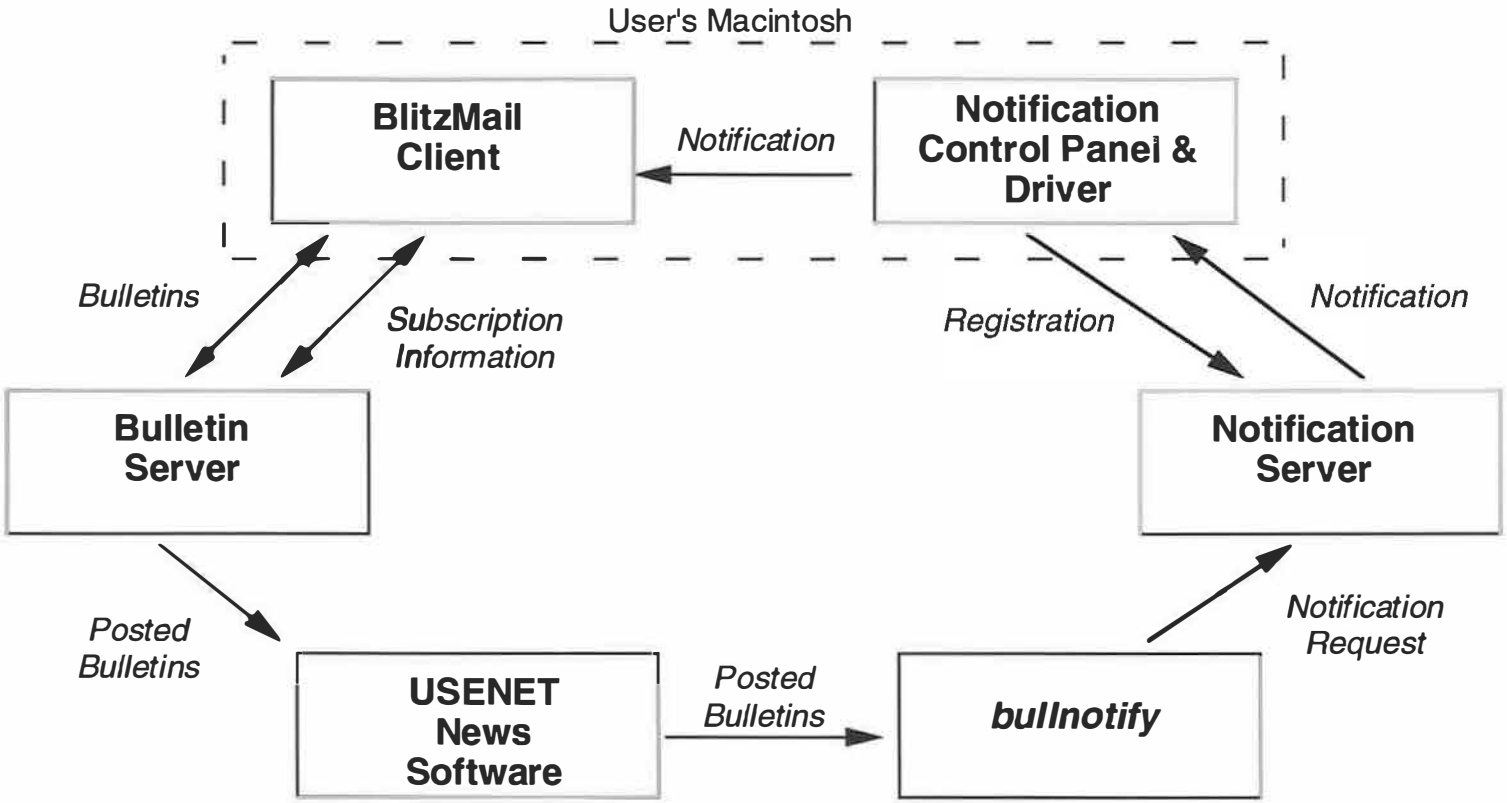


Figure 6
Dartmouth Bulletin Data Flow

in the topics that the user monitors. If there are, the client requests header information about them and presents it to the user in a format much like an email in box. If the user double clicks on a bulletin, the client retrieves the entire article.

The bulletin server also knows who the topic moderators are. If the BlitzMail user is a moderator of a group, then they are allowed to compose a new bulletin for that group. The client sends the posting to the bulletin server using the NNTP POST command. The news software is configured so that all postings to bulletin newsgroups are also sent to a program called *bullnotify*. *Bullnotify* maintains a list of users who are monitoring each topic and sends a request to the notification server to notify those people's BlitzMail clients of the arrival of a new bulletin. The notification mechanism is the same as that for newly arrived mail messages.

The Notification Server

The notification server provides a flexible mechanism for asynchronously notifying a Macintosh user of some event. This paper has already described two applications: notification of the arrival of new mail and notification of the posting of bulletins. When the user turns on their Mac, the notification control panel registers with the notification server by sending it a message that says "this person is using a Mac at this network address." An application requests notification by sending the notification server a message containing the user's name, an indication of what type of event has happened, and some optional text related to the event, for example the subject of the new mail message. If the notification server has an active registration for the user, it sends a message to the notification driver on that Mac. If the person is not registered, the notification server saves the notice and will send the notification message later when the user registers.

The user can select how to be notified about each type of event. Possibilities include a beep, a flashing icon, a dialog box, or the automatic opening of a window for reading the new message. To accomplish the latter, the reading application such as the BlitzMail client polls the notification driver and initiates the reading operation.

Mail by Name

One of the keys to a successful email system is simple addressing. A person's email address should be intuitively obvious; it should not be necessary to know arbitrary account names and host names. To send someone email, it should be sufficient to know the person's name and the name of their institution. At Dartmouth we have accomplished this addressing-by-name by using the Dartmouth Name Directory in our mail system.

As previously described, the BlitzMail system allows a person's name to be their address; the BlitzMail server looks the name up in the DND and finds their preferred local email address, their "real" email address. We have provided addressing-by-name for users of other email systems at Dartmouth by creating a mail hub that also can look up people in the DND. This allows mail to be addressed to *name*@Dartmouth.EDU with exactly the same semantics for *name* as applies when a BlitzMail user addresses a message by typing the recipient's name.

The mail hub is a Unix machine running the *sendmail* mail forwarding agent. The hub receives all email addressed to *localpart*@Dartmouth.EDU. Most sendmail machines handle mail addressed to the local machine by first seeing if *localpart* is a mail alias and, if not, assuming that it is a user mailbox on the local machine. The slightly modified sendmail on the Dartmouth mail hub also first checks *localpart* to see if it is an alias, but if it is not, *sendmail* looks up *localpart* in the DND and reroutes the message to the preferred email address specified in the DND entry. If the preferred address is BlitzMail, then the hub retrieves the name of the addressee's BlitzMail server and forwards the message to the server. If the preferred address is an RFC822 address, the hub forwards the message in the conventional way. If *localpart* is not found in the DND or if it generates more than one match, then the mail message is rejected.

Email on Paper

Even at Dartmouth some people do not have personal computers or for other reasons do not receive email electronically. But everyone can receive conventional printed mail through campus mailboxes, called "Hinman boxes" after the name of the campus post office.

Once again the Dartmouth Name Directory is the key to reaching these people since everyone has an entry in the DND — even non-computer users — and one field of that entry is the person's Hinman box number. We created a special domain, Hinman.Dartmouth.EDU, to address Hinman boxes. If the person cannot receive email electronically (or chooses not to), their DND preferred email address can be set to *name*@Hinman.Dartmouth.EDU. On the mail hub, *sendmail* directs messages for the Hinman domain to a special program that formats email messages appropriately for sending through the Hinman mail system. The resulting messages are printed several times a day, folded, and given to the Hinman post office for delivery, usually within 24 hours.

Tying It All Together

Having examined the components of the Dartmouth electronic mail system, we can summarize the characteristics that combine to create a successful system.

- The Macintosh is the preferred personal computer for the vast majority of people at Dartmouth. For 10 years it has offered a superior user interface, and many applications are now available both in house and commercially. The machine itself is moderately priced, with the recommended system still costing less than $1,200.

- The native Macintosh networking protocol suite, AppleTalk, lets a user take a new machine, preloaded with Dartmouth software, out of the box, plug it into a network jack anywhere on campus, and immediately start using network services. No network address administration is required for individual machines. Network components such as routers are relatively inexpensive.

- We have blended the AppleTalk network and the IP network almost seamlessly. A Macintosh client can easily use a server on the IP network.

- A single user directory containing *all* members of the institution is available as a resource for any network application. The directory contains the information needed to route email to a person, and people can easily change the information themselves.

- We have a Macintosh email client, BlitzMail, with an easy-to-use interface and full functionality.

- The notification system provides the proper feedback and reinforcement to users. As good as BlitzMail is, it still takes time to "check your mail." The notification system informs users exactly when it is useful to do so.

- A central mail hub uses the name directory to route messages among the Macintosh mail server, other campus mail machines, and the Internet. Mail from any source addressed to *name*@Dartmouth.EDU gets rerouted to the person's preferred address, including their campus paper mailbox if they do not use email.

Email and the Institution

A tool as powerful as email inevitably has an impact on the institution in which it is used, whether that institution is a school, a business, or a household. While it has enormous benefits, it also makes demands on the institution, raises ethical and legal issues, and often causes unexpected changes in the way the institution works.

Staffing

An email system requires people to support it. User support requires people to answer users' questions by email, by phone, and face-to-face. It requires people to plan and carry out training, especially if a large portion of the user base is not technically experienced. It requires someone to produce and distribute documentation and someone to provide postmaster services. Dartmouth has a staff of student consultants and two full-time equivalent employees who answer questions, a staff of two to manage training, and a staff of two to handle documentation.

A large component of administrative support is management of the name directory. This includes obtaining regular updates from the registrar and personnel offices as well as handling individual requests for new entries or special changes to existing ones. Dartmouth has two people who spend part of their time providing this support.

Technical support includes operation, system administration, and maintenance of the servers and the network, repair of personal computers, software development, dealing with vendors, and planning for growth. Dartmouth has two system administrators who spend part of their time on the email servers, two network maintenance technicians, four software developers, and two people who spend part of their time on planning.

Management support includes establishing and enforcing email policies, maintaining good relations with other parts of the institution, and managing finances. The Computing Services organization at Dartmouth has three managers who are involved in these tasks.

Email Rights

An email system raises ethical and legal issues that may be new to the institution. How private are email messages? Does management have the right to read people's email? Do postmasters? System administrators? What if the police ask to see someone's email as part of an investigation? Institutions need to have clear written policies about email privacy. Dartmouth has established a Computing Code that spells out everyone's computing rights and responsibilities, and no one may intentionally read someone else's mail unless there is strong evidence that there has been a violation of the Code or of the law. But these occurrences are rare. What is more common is that email messages get incorrectly addressed or run into technical problems in the mail system and end up being seen by an unintended recipient or by the postmaster. For this reason we emphasize to users that they should not send anything through the email system that they would not want strangers to see. We tell them that the privacy of an email message is between that of a first class letter and a postcard.

What constitutes appropriate use of the email system? How do you handle unsolicited and possibly unwanted messages? How do you handle impersonations and harassment by email? What about using the email system for personal communications? One solution to unsolicited messages is to create a channel for the information so that only those who want to receive it do so. Mailing lists, special newsgroups, and Dartmouth Bulletins provide those channels. Dartmouth handles harassment cases — whether by email or not — through the deans' offices. Members of the college community are welcome to use the email system for personal activities as long as those activities meet accepted standards of ethics and etiquette. For example, people may use the college's email system to participate in mailing lists and newsgroups so long as they observe good manners and the conventions of the group.

In all cases where the use of email raises ethical and legal issues, the institution must guarantee due process to those involved. People on all sides of the issue must know what the rules are, who enforces them, and what the appeals process is. All this should be written and presented to the user when they become a user of the system. Dartmouth provides this information in the packet given to entering students and new faculty and staff.

The Cultural Impact of Email

Email changes the way people conduct their business, but the changes are difficult to quantify. This is a rich field for study. Two anecdotal cases from Dartmouth's experience will illustrate.

Email can be intrusive. The mechanisms we have designed to notify us of the arrival of new messages are as intrusive as that other modern invention, the telephone. Students, faculty, and staff have all reported that if they really need to get some work done, they have to escape not only the phone, but the email system as well. They find it difficult to resist the temptation to check for new mail.

Email breaks down the mechanisms that organizations have built up to control access to people. You can send email to a subordinate, a peer, or the CEO with equal ease. Many people who do not answer their own telephones do read their own email. Answering machines, secretarial staffs, and scheduled office hours no longer provide "protection."

What's Ahead?

Dartmouth College has built a successful campus email system — one that continues to evolve. One of the questions confronting us now is how best to support multi-media and privacy-enhanced email. Another is how to provide email on a network with increased use of dialup and mobile addresses. And finally, as components of our mail system "age," how do we (belatedly) perform life-cycle planning and retire old systems without interrupting the email service that our campus community has come to expect?

Acknowledgments

The Dartmouth email system is the product of many talented people. Rich E. Brown, David Gelhar, and Jim Matthews are the designers of BlitzMail and the Dartmouth Name Directory. Dartmouth Bulletins were conceived by Nancy Hossfeld and Randy Spydell and implemented by Jim Matthews, Pete Schmitt, and Stephen Campbell. Stan Dunten is the architect of the AppleTalk network, and Philip Koch designed and implemented the AT/IP gateway. Stephen Campbell implemented the mail hub and the Hinman mail system. Nancy Hossfeld and Rich Brown guided the creation of Dartmouth's email policies and has documented the system. Punch Taylor provided technical oversight throughout.

Availability

Dartmouth College licenses the BlitzMail mail system. Corporate and educational/non-profit licenses are available. Contact Software Sales, Dartmouth College, 6028 Kiewit Computation Center, Hanover, NH 03755-3523. Phone 603-646-2643 or fax 603-646-2810. Details are also available on the Internet from ftp.Dartmouth.EDU or gopher.Dartmouth.EDU.

The Dartmouth AT/IP gateway code is licensed to Engage Communications Corporation of Aptos, CA.

Bibliography

Specifications and other details about components of the Dartmouth mail system can be found in the following documents. Dartmouth documents are available by contacting Consultants' Office, 6028 Kiewit Computation Center, Dartmouth College, Hanover, NH 03755-3523, phone 603-646-3417.

Richard E. Brown, "The Kiewit Network: A Large AppleTalk Internetwork," Proceedings of the ACM SIGCOMM '87 Workshop, ACM Press. Provides a detailed description of the AppleTalk network, its components and protocols.

Dartmouth Computing Services, "BlitzMail Primer and Reference Manual," August 1993. Everything the user needs to know about BlitzMail.

David Gelhar, "The BlitzMail Protocol," 1994, internal manual. Detailed description of the BlitzMail server and the client/server protocol.

David Gelhar, "BlitzMail System Manager's Guide," internal manual. Describes the BlitzMail servers from a system administration point of view.

David Gelhar, "Protocol Specification: DND Server Interface," 1993, internal manual. Describes the DND server protocol, the DND fields, and the name matching algorithm.

Philip D. L. Koch, "The Dartmouth AT/IP Gateway," 1992, internal manual. Describes the functionality and command and configuration language of the AT/IP gateway.

Internet Information Resources for the System Administrator

Thomas Barrett

Pacific Bell

tjbarre@srv.PacBell.COM

The advent of the Internet is perhaps the most important development in computing in the last decade. The widespread connectivity that this network and its attendant protocols (TCP, UDP, IP) have fostered has resulted in a level of computer communications previously undreamed of.

For many system administrators however this has only had the effect of increasing the complexity of mail administration, and of insuring that crackers from remote sites are denied access to local systems, while at the same time local users who need Internet access must be provided with it. Thus the Internet can be something of a headache for the system administrator.

The Internet, though, can also be a rich source of helpful tools for the system administrator who knows where (and how) to look. In this paper we will investigate a number of utilities that can assist the administrator with his or her daily job, and which are readily available from various archives on the Internet. It is assumed that the reader knows how to retrieve files over the Net--for more details on this see "Internet Discovery and Retrieval Tools" by Amy Kreiling elsewhere in these Proceedings.

Archive Sites

There are many archive sites on the Internet, which, in total, make hundreds of Gigabytes of files available to the general public through anonymous ftp, gopher, mosaic, fsp, and other mechanisms. In these sites you can find many tools to make your job easier. But where are these sites?

If you use gopher or mosaic to access the Internet you have a "built-in" list of sites available to you since these utilities "know" what other sites are out there. If you use a more basic utility though, such as ftp, you must know where you are going! Many of the archive sites are mentioned in the USENET newsgroup alt.internet.services--in particular Scott Yanoff periodically publishes a list of Internet services available. Regular readers of this group will discover the locations of many different Internet archives and facilities.

Another good place to start is at the major archive sites in the United States. These are wuarchive.wustl.edu at Washington University of Saint Louis, gatekeeper.dec.com (Digital Equipment Corporation), archive.cis.ohio-state.edu (Ohio State University), and ftp.uu.net (UUNET Technologies, Inc. in Falls Church, Virginia). If a useful utility is out there, one of these sites will probably have a copy.

There is a fairly comprehensive listing of guides and tutorials to various Internet utilities maintained by John Arthur December on ftp.rpi.edu. This is available by anonymous ftp in the directory pub/communcations. The file is named internet-cmc.ps.Z or internet-cmc.txt (for postscript or ascii versions, respectively.)

Finally, users with access to archie can query for the utility desired and archie will provide them with its exact location.

Types of Utilities

The job of system administration is a varied one, and the administrator has many different job functions

to perform. Consequently there are many different types of utilities that may prove to be of assistance to the administrator. In this paper we will discuss mail utilities, time programs, scripting languages, security utilities and packages, and Internet firewall programs.

Mail Utilities

There are a number of public domain utilities that can make life easier for you and your users when dealing with electronic mail. These are discussed below.

Elm

Elm is an intelligent mail "user agent" which can be quite handy for your users, especially if they otherwise have only the System V "mail" command. Even the BSD "mail" command, although it is more advanced, still leaves much to be desired. Elm displays messages in a format similar to that of the BSD mail command, but with more information shown. It allows the users to move back and forth (actually up and down) between messages using the "j" and "k" keys as in "vi". Elm allows all the standard functions, such as replies, group replies, mail a message, forward a message, delete a message, etc. In addition elm allows the user to pipe the current message or all tagged messages to a system command, scan the current message for calendar entries, and save (or copy) the current message or all tagged messages to a folder. Folder management is one of elm's strong points, as it will automatically save messages from different users to different folders, based on the originator's login id. This allows your users to easily keep track of their correspondence. Much nicer than storing all saved mail in a single "mbox" file as the other two mail commands do.

Vacation

This is a rather handy little utility that allows the user to notify others that he or she is not currently reading e-mail. Note that although vacation is currently shipped on some platforms, such as the Sun and HP, older machines and other types of machines will not have this utility available--thus we must retrieve it from the Net. Vacation is actually fairly simple to use--you add an entry in your .forward file as follows:

```
tjbarre "|/usr/ucb/vacation -a tjb tjbarre"
```

The first field is your login name, the second an invocation of the vacation program which will respond to messages for the login "tjbarre" as well as to those for an alias, "tjb". Vacation will respond with whatever text is in the file ~/.vacation.msg. This file must be complete including all relevant header information. Thus, for this week I might construct the following .vacation.msg:

```
From: tjbarre@srv.PacBell.COM (Tom Barrett)
Subject: Your E-Mail
Precedence: bulk

I am at the SANS-III conference in Washington, D.C. through
April 6 and will not be reading my e-mail during this time. If
you have an urgent problem contact Aijaz Asif at saasif@srv.PacBell.COM
or leave voice mail on (510) 823-1941.
Thanks,
tjb
```

Note that vacation is smart enough not to respond to any messages whose precedence is "bulk" or "junk". Also, vacation will only respond if either the From: or Cc: lines of the mail header contain the specified login string or one of the specified aliases. Additionally, vacation keeps track of those senders it has responded to and will respond to each only once per week. The senders are stored in an ndbm database consisting of the files ~/.vacation.pag and ~/.vacation.dir. You must run "vacation -i" before attempting to use the vacation program to initialize these two files.

Procmail

So far we have surveyed utilities that will make your users' lives easier. What about looking at one that will make your life easier as an administrator? The procmail utility, developed by Stephen R. van den Berg of the Netherlands allows you to sort incoming mail into separate folders and files, to preprocess mail, to start programs on mail arrival, and to selectively forward incoming messages. The formail utility which is part of procmail enables generation of autoreplies and manipulation of digests and mailboxes.

While some of these overlap with elm capabilities, procmail is in fact a far more complex and feature-rich utility. The feature summary accompanying procmail stresses the following advantages:

- small code size
- ease of installation
- easy to maintain and configure because it consists of only one executable and a corresponding ".rc" file
- uses no temporary files
- filters, delivers, and formats mail reliably
- Allows you to choose whether to bounce undeliverable mail
- performs reliable mailbox locking across NFS
- provides a mail log file which summarizes where mail came from, its subject, length, etc.
- Displays diagnostic and error messages based on the above log file
- No limits on line lengths, mail length, or the use of any character. Even '\0 ' is permitted.
- Works with any RFC 822 compliant mailer

procmail can be invoked through a user's .forward file or if the administrator installs it, through the mail system itself. It is sophisticated enough to provide almost all of the functionality of elm and vacation, as well as allowing the administrator to keep tabs on what's going on with a particular system's e-mail traffic. In addition, its automatic "filing" of received messages into the appropriate folders can be of great help in insuring that users' mail files do not grow too large, especially when they are on vacation. (Note that the vacation program mentioned above does not address this problem, it merely notifies senders that a recipient is not reading e-mail.)

Directives in the .procmailrc file are called "recipes" and instruct procmail what to do with particular messages. A simple example would be:

```
:0
^To:.*black-powder@cinnamon.com
blackpowder
```

This directs procmail to save all messages from the black-powder mailing list (a list that deals with muzzle-loading firearms) to the mailfolder "blackpowder", using blackpowder.lock as the local lockfile.

An extensive set of manual pages is provided with procmail, along with some examples of .procmailrc files and many recipes. Those interested may obtain the program from ftp.uu.net in /usenet/comp.sources.misc/volume38/procmail.

Time Programs

Several programs are available on the Net to manage the clocks of a group of machines, keeping them synchronized to Greenwich Mean Time. We will discuss two of them here.

Ntp and xntp are both programs which may (and in fact should) be used to keep all of your machines synchronized to the same time value, and to an external time feed such as a radio clock. This can be important for a number of

reasons--for instance replay attacks (in which a cracker captures packets and simply "replays" them at the target machine at a later time) are frequently guarded against through the use of timestamps within the packets. If a packet is received with a "stale" timestamp, it is discarded. In order for this scheme to work, however, both sending and receiving machines must have a fairly close notion of the current time. ntp and xntp will provide the needed synchronization.

Perhaps on a more pragmatic note, it should be mentioned that in the past crackers have gone unpunished because of time discrepancies between machines. At SANS-II Randy Marchany of Virginia Tech mentioned a security incident in which his attorneys decided not to use the system logs as evidence because of the differences in timestamps. Thus we have another reason for keeping all machines synchronized to a reliable time feed. Again ntp and xntp are the tools for this job.

NTP stands for Network Time Protocol, and was originally introduced in Internet RFC 1059. The ntp utility implements version 1 NTP as specified in this document. xntp is an implementation of the version 2 Network Time Protocol as set forth in RFC 1119. The purpose of these utilities is to enable a group of machines to synchronize as closely as possible to UTC as determined by the best time source available. This source can be specialized hardware (such as a radio clock, mentioned above), or another machine, a timeserver. Synchronization proceeds through a hierarchy of timeservers (called a synchronization subnet) with servers classified as to how far they are from an external time source. Those with direct access are stratum 1 servers, those served by a stratum 1 server are stratum 2 servers, etc. The maximum number of strata is limited to 15--far more than should ever actually be necessary.

The ntp and xntp utilities use /etc/ntp.conf as their configuration file. In this file the administrator can specify what machines to poll with time queries. If you have a machine that is directly connected to an external time source, you can specify this in the /etc/ntp.conf file as well, using an entry of the form 127.127.t.u where t is the type of clock (actually the clock driver) and u is a unit number whose meaning is dependent on the value of t. All clock entries begin with 127.127. As an example, the driver for Precision Standard Time 1010(WWV)/1020(WWVH) is driver number 3--thus we might specify 127.127.3.1 as the external time source for a stratum 1 host with such a clock.

The /etc/ntp.conf file is also where you specify the name of the system's "driftfile". The driftfile keeps track of how accurate (or inaccurate) the system's clock is--normally this takes a day or so for the ntp utilities to determine. The sole purpose of the driftfile is to avoid going through this synchronization period again if the daemon is stopped and re-started.

Address and mask restrictions in the /etc/ntp.conf file allow the utilities to restrict the privileges of remote machines. For example the following statement:

```
restrict default notrust nomodify
```

tells the local system that by default it should not trust any remote machines, nor should it allow them to modify its time value. This is a particularly sensible default to use, since many many machines will be able to contact you with ntp or xntp packets on the Internet. This statement can then be followed up by statements granting various privileges to known or trusted hosts.

To insure the identity of time peers or servers xntp allows the use of an authentication facility, which adds a 32 bit key ID and a 64 bit checksum to each packet. The sending and receiving machines must share a set of keys and key identifiers. The receiving machine recomputes the checksum using a DES algorithm with the key specified by the key ID in the received packet--if the value matches the received checksum all is well--if not the packet is discarded.

Scripting Languages

There are a number of scripting languages in the public domain that are available over the Internet. We will discuss

two of the most popular here.

Tcl and Tk

Tcl (pronounced "tickle") and Tk were developed by Professor John Ousterhout of the Computer Science Department at the University of California, Berkeley. Tcl stands for "tool command language" and is essentially a shell- or awk-like programming language, which is distinguished by its ease of incorporation into application programs, allowing those programs to use the additional routines of the Tcl "core". Because of this, Tcl is ideally suited to rapid prototyping. The combination of Tcl and the Tk toolkit allows for rapid and extremely simple GUI development. Tcl can be invoked interactively from the "tickle" shell, tclsh. Tk can be similarly invoked from the windowing shell, wish. While a complete exposition of Tcl and Tk is beyond the scope of this paper, we will give a few programming examples.

First, here is a simple Tcl program to print out "hello world!":

```
#!/usr/local/bin/tclsh
set a "hello world!"
```

This is certainly one of the simplest "hello world!" programs you're ever likely to see. As it happens the set command in Tcl is used both to set and print out values of variables. The printing of "hello world!" in this example is just a side effect of setting the value of the variable a.

Let's try something a bit more ambitious, and also a bit more useful to a system administrator. Here is a Tcl program to report those filesystems with 5000 blocks or less free space:

```
#!/usr/local/bin/tclsh
set a [split [exec df] '\n']
set L [llength $a]
for {set i 1} {$i < $L} {incr i} {
        set b [lindex $a $i]
        set space [lindex $b 3]
        set filesystem [lindex $b 0]
        if {$space < 5000} {
                puts "$filesystem is down to $space blocks"
        }
}
```

The above example illustrates a number of Tcl concepts, particularly with regard to list management. split does exactly what its name implies, it splits a list into separate words at the character specified, in this case newline. llength returns the length of a list. lindex returns a particular member of a list, the index being provided by the second argument. This example also shows some of the control structures available in Tcl, such as the "for" loop and the "if" conditional.

Perhaps of more interest in today's computing environment is the GUI building capability of Tk. Here is a simple example of a Tk script:

```
#!/usr/local/bin/wish -f
button .b -text "hello world!" -fg white -bg black  -command exit
pack append . .b top
```

This simple sequence of commands creates a button widget labelled with the text "hello world!" in white on a black background, which, when depressed, exits wish. pack is the geometry manager used to realize the widget. In general widget creation and realization are extremely simple with Tk, far more so than with packages such as the Xt toolkit of Xl 1. The complex part of the code becomes the Tcl event handlers.

It should be mentioned that there is an extension to Tk called XF which actually provides an interactive GUI builder. This makes it even easier to construct a GUI.

For those with a deeper interest in this language I heartily recommend Professor Ousterhout's book "Tcl and the Tk Toolkit".

Perl

Perl is an acronym for Practical Extraction and Report Language. It is a programming language developed by Larry Wall of the Free Software Foundation. It combines features of C, sed, awk, and sh. It is an interpreted language, like Tcl, and is primarily intended for text manipulation and report generation (hence its name). It is able to deal with arbitrarily large strings, arrays, and levels of recursion subject only to memory constraints on your machine. Additionally, perl can provide you with better performance than sed, awk, or sh.

Perl programs allow you to use the C preprocessor if you desire; however perl comments begin with '#' so you must be careful not to begin a comment with a word recognized as a cpp directive.

Perl recognizes three basic datatypes--scalars, arrays of scalars, and associative arrays of scalars. Arrays are indexed by number while associative arrays are indexed by string. Certain of the operations in perl are "context-sensitive". That is, they might returna an array if the context warrants, otherwise they will return a scalar. Similarly scalar variables and values may be interpreted as either strings or numbers depending on the context.

Scalar variables are referenced using the '$' character, just as in the Bourne shell--thus $s refers to the value of scalar variable s. Arrays are referenced using the '@' character so @a refers to array a. Associative arrays are referenced by the '%' sign so %aa refers to associative array aa.

Assignments can be made both to scalar variables and whole arrays (of either type) or to array "slices". The type of the lvalue in an assignment dictates the context for the righthand side of the expression.

As with Tcl and Tk, a complete exposition of perl is beyond the scope of this paper. We will however give a few example programs. First, here is a simple perl program to print out "hello world!":

```
#!/usr/bin/perl
print "hello world!\n";
```

This is again a fairly simple program, and shows perl's basic similarity to the shell. Now let's try something a bit more ambitious. Here is an example of a scalar array as an lvalue:

```
#!/usr/bin/perl
($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = gmtime(time);
print "GMT is now $hour:$min:$sec\n";
($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
print "local time is now $hour:$min:$sec\n";
```

This example prints out the current time in both GMT and local formats. Notice that the array elements can be re-used as lvalues.

The manual pages provided with the perl package are fairly comprehensive; there are also a number of publications out for those with a deeper interest in perl.

Security Utilities and Packages

There are a number of security packages available on the Internet. We will examine three of the most popular.

Crack

Crack is a facility that allows you to check the "goodness" of your users' password selection. Goodness in this context equates to difficulty of decryption.

Crack was developed by Alec David Edward Muffett. Essentially crack takes a dictionary and a password file supplied to it by the user and attempts to find password matches based on information in the dictionary, the GECOS field of the password file, and some rules for manipulating the dictionary and GECOS entries supplied to it in the files dicts.rules and gecos.rules. These rules deal with the various permutations of the dictionary and GECOS words that should be tried--for instance a rule might specify appending the current word to itself. There is a fairly simple language supplied for writing new rules, but in general the rules supplied with crack should prove sufficiently exhaustive to provide a good password check.

Crack is essential if you run a UNIX that allows poor password choices, as it allows you to detect those choices and to notify the offending users. Unfortunately it is fairly CPU intensive and takes a fair amount of time to run. It probably should not be run on a production machine.

Beginning with version 4.1 crack uses Michael Glad's UFC crypt() (ultra fast crypt) and so the encryption routines run a bit faster. On the other hand the rules for manipulating the GECOS information have become a bit more thorough, so there are more potential passwords to try--this balances out any gain made by the use of UFC crypt, but provides more complete password checking.

COPS

COPS is a collection of security tools intended to assure that all is right with your system. Its purpose is primarily one of prevention--COPS will note and report on most of the common security holes that may inadvertently have been left on a system.

COPS is an acronym for Computer Oracle and Password System. It is actually a suite of 15 programs and is designed to perform the following security checks:

- Check for "dangerous" permissions (world-writable or world-readable) on the vital system files and directories listed in is_able.lst (this is a configuration file that lets the individual administrator specify which files and directories are "vital".)

- Check for inappropriate permissions on device files (i.e. world-readable or world-writable filesystems.)

- Check suid status of all files in the system, notifying the administrator of any changes in status. Also flag those suid files which are world-writable, are shell scripts, or are not executable.

- Check the /etc/passwd file and NIS database for null passwords, improperly formed or blank lines in the password file, non-unique uid's, non-numeric group id's and non-alphanumeric user-id's.

- Check the /etc/group file and NIS database for improperly formed file entries, blank lines, and non-unique group id's.

- Check for "goodness" of passwords. COPS runs a very small subset of the attempted passwords that crack would try based on the login id and GECOS fields of the password file. Note that COPS is no substitute for crack, which is a far more comprehensive password tester.

- Check to insure that root's PATH and umask are set correctly. Also check that root owns /bin, /etc, /etc/passwd, /.login, /.profile, and /.rhosts.

- Check the /etc/rc* files to ensure that none of the files or PATHS used are world-writable.

- Perform the same check on the files in /usr/lib/crontab.

- Check the users' home directories to insure that none are world-writable.

- Check that important user files such as .rhosts, .profile, etc. are not world-writable.

- Check for correct ftp set-up, if appropriate.

- Check for changes in CRC values generated from system files. If a value changes the file has been corrupted or tampered with. This is a *very* handy utility.

- Check for miscellaneous problems--these include world writable programs in /etc/inetd.conf or /etc/servers, uudecode alias in the mail alias file, etc.

- If given a goal (compromise root) and a list of user and group id's that can be used in the attempt, COPS will search through the system until it can determine whether the goal is attainable or not. This last utility comes from Robert Baldwin's U-Kuang System and is the only part of COPS that is not in the public domain.

COPS should be run daily by cron, at periods of low CPU load. Used in this fashion it provides the benefit of nearly continuous security monitoring--something most system administrators do not have time for.

Tripwire

Tripwire is a file integrity checker authored by Gene Kim and Eugene Spafford of Purdue University. Its goal is to spot changes, whether due to filesystem corruption or malicious tampering, in a designated set of files. While it would appear to overlap somewhat with the file integrity testing capability supplied by COPS, Tripwire is actually far more thorough.

Tripwire is fairly simple in concept. The administrator first specifies a set of important files and directories which should be checked in the file tw.config. There is a set of "cpp-like" directives that can be used to tailor the tw.config file to a particular site. These directives allow the checking of different sets of files based on hostname. This makes it easy to maintain a single tw.config file which can be used for multiple hosts.

Next, tripwire -initialize can be used to construct the database containing the initial file integrity information. At this point Tripwire will detect any changes to the specified files. Tripwire cannot, however, assure you that the files used to generate the initial database are "clean". The only way to be sure of this is to reinstall all system binaries before running tripwire -initialize.

Once a database has been set up, Tripwire can be run in integrity checking mode by invoking it with no arguments. Tripwire's integrity checks consist of spotting added or deleted files, and comparing each file's signature to that stored in the database. There are 7 different signature generation routines provided with Tripwire: MD5, Snefru, MD4, MD2, SHS, CRC-32 and CRC-16. The first five of these are message-digest algorithms or one-way hash functions. The last two are the familiar cyclic redundancy checks with either a 32 or 16 bit signature space. These are provided for speed should the administrator wish to run Tripwire fairly frequently (say, every hour). Unfortunately the CRC routines are easily cracked, so the stronger message-digest routines should be run at least once a day. MD5, MD4, and MD2 come from RSA Data Security, Inc. Of these MD5 is the most advanced. MD4 is less secure but about twice as fast. MD2 was developed for use with Privacy Enhanced Mail and is extremely slow compared to the other two routines. MD2 is not in the public domain, although arrangements have been made for its distribution with Tripwire in its current form. All three of these RSA routines generate 128 byte signatures, making them very difficult to crack. Snefru is the Xerox Secure Hash Function. It has never been broken in its 4-pass form (the form shipped with Tripwire). SHS is an NIST proposed Secure Hash Standard. It is possible, and indeed the default behavior of Tripwire, to construct a database with multiple signatures in it. Checking of multiple signatures, especially

multiple strong signatures, virtually guarantees file tampering will be detected. By default Tripwire calculates the MD5 and Snefru signatures for all files in the tw.config file and stores them in the database for use in integrity checks.

Tripwire provides both interactive and command line facilities for updating the database should files change legitimately, or packages be added or deleted.

FSP

FSP (File Service Protocol) is a handy server utility which many administrators may want to consider running as an alternative to (or in addition to) the more conventional FTP utility. Two of the major problems in setting up an FTP server are the amount of traffic generated, and the amount of CPU cycles consumed. FSP attempts to eliminate, or at least alleviate these problems. For instance, FSP never forks, so no additional processes are added to the system's load. Also FSP allows you to limit the amount of data the daemon will send out, thus regulating traffic.

Additional benefits of FSP are that it is essentially a state-less protocol, and not connection oriented like FTP. This allows FSP to survive failed connections--it will simply wait until the connection comes back up to continue. The state-less nature of FSP is made possible through the use of datagrams instead of the TCP sockets used by FTP.

Finally, FSP is resistant to "flooding" attacks. Each client request contains a key; If this key does not match that in the server's database for either the current or next transaction the server will not accept the request unless one minute has elapsed since the last request was accepted.

FSP comes with a number of stand alone utilities for the client that "throw" packets at the server. These have names such as fcdcmd (which performs a 'cd' on the server), flscmd (which performs an 'ls' on the server), etc. There are a number of user-friendly interfaces for use by the client though, among them fspclient which provides an FTP-like interface, or fspcli which provides similar functionality but is implemented through a fairly small PERL script. An xview client, fsptool, is also available, although this may be difficult to set up if you are not working on a SUN computer.

There is now a news group devoted to fsp, alt.comp.fsp, as well as an fsp mailing list at fsp-discussion@Germany.EU.net. The official repository for fsp is ftp.germany.eu.net in the directory /pub/network/inet/fsp.

Internet Firewall Programs

One of the most important jobs of a system administrator is keeping a clear and controlled separation between internal networks and the Internet. For although the Internet is a wonderful source of utilities, it is also an almost endless source of crackers. There are several tutorials on firewall construction available over the Internet. Some may say that if you're connected to the Internet and can retrieve these documents then you had better have constructed a serviceable firewall already. To this I would counter that it's always possible to improve upon an existing firewall. An inquiry to archie reveals a number of useful tutorials on firewalls and their construction. I would particularly recommend "Thinking about Firewalls" and "A Network Firewall" by Marcus Ranum, or "An Architectural Overview of UNIX Network Security" by Robert Reinhardt. These papers present the basic types of Firewalls available--the Screening Router, Bastion Host, Dual Homed Gateway, Screened Host Gateway, and Screened Subnet. Each of these has its own level of security balanced against more or less ease of use to the internal users. It is the job of the system administrator to balance these two concerns. The papers mentioned above can be of assistance in this task.

Conclusions

This paper has attempted to present a survey of the types of utilities available on the Internet. It must be stressed though that we have only discussed a few of the many many helpful utilities, packages, and services provided on

the Internet--the "tip of the iceberg" as it were. The best way to get acquainted with the Net and the resources it has to offer is to jump in and start browsing the archive sites. You're sure to find some tool that will help to make your job easier!

An Introduction to Internet Discovery and Retrieval Tools

Amy K. Kreiling
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
kreiling@cs.unc.edu

Abstract

The Internet is a worldwide collection of networks that contains an unimaginable amount of information -- public domain tools, news services, electronic mail, databases, even bitmapped images of people's faces. Until a few years ago, most of this information was accessed by way of electronic mail, network news or anonymous FTP. Today, a wide variety of tools exist, making access to the vast resources of the Internet even easier. This paper will present an introduction to these tools and hopefully provide the reader with the hows & whys of using these utilities. Tools such as NCSA Mosaic, Gopher, Veronica, and Archie will be covered, as well as a brief introduction to the information systems they use, such as WAIS and the World Wide Web. This paper will provide valuable information for users of all types of computers -- PCs, Macintoshes, and UNIX workstations.

Introduction

For those of us in the academic world, at the larger colleges and universities, the Internet has been a part of our world for several years. Electronic mail consumes a large part of our day, and the remainder of the day we spend reading USENET news groups. We have friends at other Internet sites with whom we electronically correspond on a regular basis; perhaps we have collaborated with remote sites on a project or two; and probably quite a few of us have, at one point or another, gotten caught up in a juicy relay chat session or an intensive MUD game. We've taken the Internet for granted and may be a bit bewildered at the current hype it's receiving by the media, government and general public. Hey, it's just a network -- what's the big deal? But it is a big deal -- practically hundreds of people are getting connected to the Internet every day, and each person brings with him or her an individual definition of the electronic frontier. All of those ideas are slowly changing the Internet with which we are most familiar. Now we "Surf the Net" or "Cruise the Information Highway" -- instead of just e-mail and news groups to cycle through each day, we also have Gopher and World Wide Web servers to access, on-line electronic libraries and museums to browse, and thousands of electronic government documents to pour through. When do we find the time to do the work for which we were originally hired? Hopefully, this paper will help you deal with the onslaught of information and introduce you to the tools and ideas that will assist you in your search for that elusive piece of information.

The Internet and its History

The Internet can be defined as the world-wide "network of networks" connected to each other using the IP protocol and other similar protocols. It's important to note that the Internet is not just the NSFNET but rather the whole collection of regional networks that connect into the larger, more formal networks. The larger and faster networks, such as the NSFNET, are called the backbone of the Internet. Networks such as Compuserv and BITNET are connected to the Internet by gateways and are considered by some not really a part of the Internet but rather existing on the fringe of the Internet. More and more though, the differences between these networks are being removed from the average user by smarter and more sophisticated gateways, creating one huge, worldwide conglomeration of a network [Krol92]. Many now refer to this as the Matrix.

In 1969 the United States Department of Defense created the ARPANET -- a research network designed to connect the military with industries, research organizations and universities doing research for the DoD. In the mid-1980s, the DoD created the MILNET for its own use, and the National Science Foundation created the NSFNET by connecting the five NSF funded supercomputer centers.* In March 1990, the ARPANET was officially disbanded, and in 1991, then Senator Al Gore introduced his "High Performance Computing Act" bill that established the National Research and Education Network. The NREN will provide more connections to small colleges, high schools and elementary schools. It is also expected to provide a faster network, but with an existing Internet user base of 20 million people and a growth in Internet traffic of 10% per month, the bigger pipes the NREN is expected to provide will support not so much faster traffic as simply more traffic.

In the Beginning

Most of us got our start on the Internet by using electronic mail. It's easy, fast and fun, and you probably can't imagine life without it. Maybe you joined a few mailing lists and eventually became so overwhelmed with e-mail that you started looking for a new way to organize your information. USENET news groups attracted your attention -- here was a way to read all of that information at your own pace on your own schedule.

Currently, there are over 2000 news groups in existence, and that number is expected to increase dramatically. Most of us have experienced the dread of returning from a vacation, faced with hundreds of unread news articles. If you were waiting for a reply to a question you posted to your favorite news group, the answer has probably already been expired. Do you face the flames of reposting your question? Luckily, most of the USENET news groups are archived and available to the Internet community through anonymous FTP. But, how do you find the anonymous FTP server that is currently archiving your favorite news group? The Internet's FTP servers are filled with public domain software for all types of computers and operating systems, archived news groups, technical papers, image and audio files, documentation on every imaginable subject, and anything else that people want to make available to the electronic community. With over

* The author first encountered the Internet at one of these centers. She became a self-professed information junkie at NCSA and continues her passion at UNC.

1200 systems allowing anonymous FTP access, how do you find the single piece of information in which you are interested? This is where our Internet discovery and access tools come into the picture.

Archie

A few years ago, some people at McGill University decided to create an indexed database of all the information they could find on anonymous FTP servers from around the world. They wrote a program that traverses the directories available via anonymous FTP and creates a database of the files and their location. Users access this database by running an Archie client. As Archie became more popular and the load on the Archie server at McGill increased, mirrored regional Archie servers were created. The information on each server is identical, but to be a considerate Internet user, you are expected to point your Archie client to the server closest to your region. Currently, Archie servers can be found at:

archie.internic.net	198.49.45.10	AT&T server, NY (USA)
archie.unl.edu	129.93.1.14	Univ. of Nebraska (USA)
archie.rutgers.edu	128.6.18.15	Rutgers University (USA)
archie.sura.net	128.167.254.195	SURAnet server MD (USA)
archie.sura.net (1526)	128.167.254.195	SURAnet alt. MD (USA)
archie.doc.ic.ac.uk	146.169.11.3	United Kingdom Server
archie.edvz.uni-linz.ac.at	140.78.3.8	Austrian Server
archie.switch.ch	130.59.1.40	Swiss Server
archie.th-darmstadt.de	130.83.22.60	German Server
archie.unipi.it	131.114.21.10	Italian Server
archie.uqam.ca	132.208.250.10	Canadian Server
archie.wide.ad.jp	133.4.3.6	Japanese Server
archie.sogang.ac.kr	163.239.1.11	Korean Server
archie.au	139.130.4.6	Australian Server

There are other Archie servers, as well -- the "servers" query will give you the addresses of Archie servers around the world.

The Archie database contains information on more than 1.2 million files. The database is searched using queries composed of a specific filename, a substring of a filename and even a description of a software package. Queries can be submitted to the Archie server by way of e-mail, TELNET, or by using a locally installed Archie client.

To send a query to Archie through e-mail, address the message to one of the Archie servers and place the commands in the body of the mail message:

mail archie@archie.sura.net
Subject:
help
whatis archie
prog archie

In this query, I'm asking Archie to send me a listing of all of the commands available for Archie access by e-mail (**help**), querying Archie's descriptive database for anything with the word "archie" in it (**whatis archie**), and requesting a listing of all of the anonymous FTP sites that have Archie clients available for downloading (**prog archie**). The Archie server will send its reply to me by e-mail.

Access to Archie by TELNET is also very simple. **Telnet** to the Archie server closest to your geographic location, login as **archie** and execute your queries. On-line help is available at every Archie server. One word of caution for TELNET access -- Archie servers are very busy, and many of the servers restrict the number of interactive users or the hours during which the server is available. Be prepared to wait or even plan to access the servers during the off-hours, such as very early in the morning or very late at night. The preferred method of interacting with Archie is via e-mail or by using an Archie client.

Clients for Archie are available for many different architectures and come in many different forms. There are clients for DOS, VMS, and NeXTStep, as well as for Unix hosts. I've also heard there is a Macintosh-based Archie client, but I have not been able to locate it, yet. The most common Archie client is the **archie** command -- you construct a single query using options to the **archie** command and wait for the results:

```
archie -s -m10 -h archie.sura.net   xarchie
```

This command will access the **archie.sura.net** server (**-h** option) and will search for the substring **xarchie** (**-s** option). The **-m10** option specifies that only the first 10 results should be returned. The information returned will look like this:

```
Host  ftp.x.org
Location: /contrib
FILE -rw-r--r--    179119  Nov 14 1991  xarchie-1.3.tar.Z
```

If this is what I'm looking for, I can retrieve the file using anonymous FTP:

```
ftp ftp.x.org
Login: anonymous
Password: kreiling@cs.unc.edu
set binary
cd /contrib
get xarchie-1.3.tar.Z
```

More complex queries can be created, as well. As this is just an introduction to Archie, I will refer you to the **archie** man page for more details.

As you may have noticed from the example above, an Archie client for X Windows also exists. **Xarchie** is available for most flavors of operating systems running X Windows. Xarchie provides a good point-and-click interface for composing queries and has an added feature of retrieving the file with just a click of the mouse.

Clients for Archie are available from anonymous FTP servers located around the world. It would be a shame to not have you put your new found knowledge to use by insisting that you use Archie to find FTP sites with Archie clients. If you need a hint, look on the FTP server **sunsite.unc.edu** in the **/pub/packages/infosystems/archie** directory.

Wide Area Information Servers

While on the subject of databases, it would seem appropriate to mention WAIS, the Wide Area Information Server. Based on the Z39.50 draft standard describing the way in which a computer requests bibliographic information from another computer, WAIS is a distributed text searching utility. A server is created by indexing every word within every document and creating a database of the index. There are over 475 registered indexed database servers. Some of the databases currently available include the Bible, Koran and other religious text, weather forecasts, recipes, Roget's Thesaurus, archives of the USENET news group, and the US zip codes. WAIS was developed in a joint effort by Dow Jones, Thinking Machines Inc., Apple Computer, and KPMC Peat Marwick. A commercial version of WAIS is available from WAIS, Inc., or you can find **FreeWAIS**, the public domain version, by using Archie.* Information about setting up your own WAIS server is beyond the scope of this document, but the distribution of FreeWAIS includes installation documentation. Also, the **comp.infosystems.wais** news group is an excellent source of information.

A user does not need to set up the WAIS server software to use a WAIS client. Just like Archie, most Internet users take advantage of WAIS by accessing a WAIS server via TELNET or by using a client installed at their site. WAIS servers available for TELNET access include **quake.think.com**, **nnsc.nsf.net**, and **sunsite.unc.edu**. Clients available for WAIS include **WAIStation** and **HyperWAIS** for the Macintosh, **xwais** for hosts running X Windows, a terminal-based client called **swais**, and many other clients for PCs, VMS, Unix, and other operating systems.

A WAIS client accesses the selected databases and searches each index for specific key words. The client then displays a list of documents containing the key words. A score is attached to each matching file, with the highest score indicating the file in which the key words occurred the most times. The WAIS client is also capable of previewing the documents from the list. This gives you the option to review the contents of the files without having to FTP them to your local system.

All of this sounds very simple, but WAIS is really a very powerful tool. When you initiate a WAIS search, you are first prompted to select the databases (referred to as "sources") you would like to search. If you do not know which sources would be appropriate, you can do a WAIS search on the "directory-of-servers" source. The directory contains information on every registered server. When searching the directory-of-servers, you should use very general keywords in order to find relevant sources. If you have your own local WAIS databases, you can add them as sources; the same is true for any remote sources you may find out about. Not all WAIS databases are registered -- you may hear about a new database while reading the **comp.infosystems.wais** news group. With a local WAIS client, you can add this new database to your list of sources.

After you have selected the sources for your search, you compose your "question." The question should contain more specific terms appropriate to the information you hope to find. When the search is completed, you'll have a list of documents containing your keywords. Viewing the documents from your list is very easy -- when accessing a WAIS server by TELNET, you simply "select" the document using the method available to the

* Another hint -- you should be able to find the latest version of FreeWAIS on **sunsite.unc.edu** in the **/pub/packages/infosystems/wais** directory.

server; using a WAIS client like xwais, it's just point and click. If you don't find what you're looking for, try the search with different key words. Having a dictionary or thesaurus handy for looking up synonyms is a good idea, too.

A local WAIS client will allow you to save your questions for future use. This is a great feature if you're setting up a WAIS client for your users. You can supply a few example questions that will give your users an idea of how to use WAIS. At our site, we have set up a question that queries a weather server for the current conditions in our area, as well as created queries used with the campus' Faculty, Staff and Student Directories.

WAIS servers and clients are still seeing quite a bit of development. To keep up with the new features and releases or simply for more information, read the **comp.infosystems.wais** news group. Also, see the **For More Information** section at the end of this document for references to Internet-based sources of information.

Gopher

Everyone has information they need to make available to their users. Typical implementations of information systems are hierarchical in nature -- the front page of a newspaper directs you to section C for the latest sports information, and section C tells you to look on page 3 for last night's basketball scores; call the bank for the latest interest rates and you'll have to press #2 for the loan department and then #4 for the current interest rates; maybe you've even written an information system for your users and used a menu-based design for users to follow to find more specific information on a topic. Gopher follows this model, only its information is found on the Internet and hopefully, set up by someone other than you!!

Gopher was developed at the University of Minnesota as a distributed campus information server. Using a menu-based interface, Gopher was designed to allow each campus organization to have control over its own data and server, yet present a unified interface to the end user. The physical location of the data and the methods used to present the information are transparent to the user -- when using Gopher, you may be looking at a menu that's on a workstation in North Carolina and with a quick key click, you could be viewing information that's served by a VMS system in New Zealand.

When you start a Gopher client, it will contact its home server for the main menu and display it after the server has sent the information to the client. As you browse through the menu and select an item of interest, the client contacts the server for information on that item. At some point, you will select a "resource" which will cause the Gopher client to retrieve a document, open a TELNET or FTP session or perform some type of search on a database or the Gopher server itself. All of this happens as if by magic, making Gopher an extremely easy tool to use.

Gopher clients are available for just about any computer system that can be connected to the Internet -- PCs, Macintoshes, workstations and large mainframe systems. Some common Gopher clients include **TurboGopher** for the Macintosh, **xgopher** for hosts using X Windows, and **gopher** for everyone else. To find the latest Gopher clients, check out **sunsite.unc.edu** in the **/pub/packages/infosystems/gopher** directory or use Archie. Access to Gopher servers is also available by way of TELNET, but I would definitely recommend installing a client local to your system.

Let's use the **sunsite.unc.edu** server for a quick test drive of Gopher:

telnet sunsite.unc.edu
Login: **gopher**

Root gopher server: sunsite.unc.edu

--> 1. About Ogphre/
2. Sun and UNC's Legal Disclaimer.
3. Surf the Net! - Archie, Libraries, Gophers, FTP Sites./
4. Internet Dog-Eared Pages (Frequently used resources)/
5. Worlds of SunSITE -- by Subject/
6. SUN Microsystems News Groups and Archives/
7. NEWS! (News, Entertainment, Weather, and Sports)/
8. UNC Information Exchange (People and Places)/
9. The UNC-CH Internet Library/
10. UNC-Gopherspace/
11. What's New on SunSITE/

Every item with a slash at the end of the line leads to another set of menus. Items with the **<?>** at the end will perform some type of search, while items without funny characters on the end or with just a period on the end will display a text file. More sophisticated Gopher clients use icons to represent the type of resource under each menu item.

If you're interested in some of the behind-the-scenes details of Gopher, use the = character on one of the Gopher menu items. For instance, on a submenu, I found this resource:

6. Rock n Roll Lyrics for searching <?>

Using the = key, I get:

Name=Rock n Roll Lyrics for searching
Type=7
Port=70
Path=waissrc:/ref.d/indexes.d/lyrics.src
Host=sunsite.unc.edu

This shows the host serving the resource, the location and name of the resource on that server, and the type of resource. This particular resource is a WAIS search of a database containing Rock & Roll lyrics.

As you use Gopher, you might find a site that has some really interesting information that you'd like to return to in the future. Most Gopher clients can create a bookmark that will save information about the location of the resource and allow you to easily access it again. For more information on bookmarks, please refer to the man page for your particular Gopher client.

Additional features of Gopher are utilities called **veronica** and **jughead**. Veronica searches an index of titles found on Gopher servers all over the Internet. Jughead, on the other hand, searches only the titles found on the local Gopher. After executing a Veronica

or Jughead search, you'll be presented with a menu built from the results of your search. This illustrates how dynamic and powerful Gopher can be. Let's continue with our **sunsite.unc.edu** example -- select #3 to "Surf the Net!" and then select #9 to try out Veronica:

```
--> 3. Surf the Net! - Archie, Libraries, Gophers, FTP Sites./
    9. Search of GopherSpace - Veronica/
```

We find ourselves at this menu:

Search of GopherSpace - Veronica

```
--> 1. Search gopherspace by veronica at NYSERNet <?>
    2. Search gopherspace by veronica at University of Cologne <?>
    3. Search Gopher Directory Titles using NYSERNet <?>
    4. Search Gopher Directory Titles using University of Cologne <?>
    5.
    6. Script to automate your local veronica menu (maltshop-0.2c)..
    7.
    8. FAQ: Frequently-Asked Questions about veronica (1993-08-23).
    9. How to compose veronica queries.
```

If we select #2, we can search the Internet Gopherspace, as seen by the University of Cologne. Veronica is like Archie in that a site on the Internet creates an indexed database of all of the Gopher sites they find and then makes that index available to the Internet. This doesn't mean that every Gopher site will be included, but it will probably include enough sites as to create useful search results. Selecting #2 gives us this (I have already entered our search keyword of "veronica"):

```
+----------------Search gopherspace by veronica at Cologne---------------------+
|                                                                              |
| Words to search for  veronica_____________                                   |
|                                                                              |
|                          [Cancel ^G] [Accept - Enter]                        |
|                                                                              |
+------------------------------------------------------------------------------+
```

After the search is finished, we get a new menu made up of items found during the search:

Search gopherspace by veronica at University of Cologne: veronica

```
--> 1. How to Compose Veronica Queries.
    2. Veronica Frequently Asked Questions (8 Oct 93).
    3. How to Compose Veronica Queries.
    4. Veronica Frequently Asked Questions (8 Oct 93).
    5. Veronica Gopher Search Engines/
    6. Veronica Gopher Search Engines/
    7. Use Veronica to Search all Gophers /
```

8. Search titles in Gopherspace using veronica/
9. Intro Veronica.
10. Exploring with Veronica.
11. Status report on Veronica development..
12. Re: Keep gopher/veronica space open.
13. Re: Keep gopher/veronica space open.
14. veronica FAQ, part 1/3.
15. WAIS Boolean veronica searches at Nevada.
16. WAIS based Veronica (included).
17. RE: Boolian searches for gopher and veronica.
18. Stopping veronica indexers.

Press ? for Help, q to Quit, u to go up a menu Page: 1/12

Notice the page count? We found 12 pages of menus and resources on the topic of Veronica! That's a pretty useful tool.

At this point, I'll encourage you to find your own Gopher client and start exploring all those Gopher servers. For additional information, see the **For More Information** section at the end of this document for references to Internet-based sources of information.

WWW and Mosaic

The World Wide Web is a distributed hypertext-based information system. Instead of presenting its information in a series of menus, WWW connects to resources by using hyperlinks found in the document. WWW was developed by CERN, the European Particle Physics Lab, and its popularity as an information server has really exploded in the past few months. In June of 1993, about 100 Web servers were known to the Internet community; in a survey taken on December 13, 1993, 623 WWW servers were found [Gray93]. That number continues to grow by exponential factors. Most of the wide-spread popularity of the Web can be attributed to NCSA's Web browser, **Mosaic**.

Available for Macintoshes, PCs running Microsoft Windows and hosts using X Windows, NCSA's Mosaic makes cruising the Internet easy, fun and very addictive. As long as you have an Internet connection, Mosaic provides a point & click interface for the Web's hyperlinked world. Mosaic clients can be found on **ftp.ncsa.uiuc.edu** in the **/Web** directory and on **sunsite.unc.edu** in the **/pub/packages/infosystems/mosaic** directory.

A few WWW servers are available for access by TELNET -- **sunsite.unc.edu** and **info.cern.ch** are two such servers. Just as for Gopher, I highly recommend installing a local WWW browser. A very popular terminal-based browser is **lynx**; other browsers include **Chimera** and **Viola** for X Windows and **Cello** for PCs running DOS. These can be found on **sunsite.unc.edu** or by using Archie.

For an example of a hypertext document and how it is presented by a WWW server, let's use **sunsite.unc.edu** one more time. SunSITE is using lynx for it's publicly accessible WWW browser:

telnet sunsite.unc.edu
Login: **lynx**

University of North Carolina at Chapel Hill home page (p1 of 4)

WELCOME TO UNC AT CHAPEL HILL!

Sun Feb 20 01:38:26 EST 1994

This is the University of North Carolina at Chapel Hill's world-wide multimedia information service. It is designed to be read with programs called [1]NCSA Mosaic (an interface to the [2]World-Wide Web) to provide any user with access to multimedia information services - with text, sound, images, and movies! Select any highlighted or underlined words in this paragraph for extended help.

Thanks to Cisco Inc. and Sun Microsystems, SunSITE now has its own T1 connection to the Net. Enjoy the speed. :)

-- press space for more, use arrow keys to move, '?' for help, 'q' to quit Arrow keys: Up and Down to move. Right to follow a link; Left to go back. H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list --

This is the text-only UNC home page; the two underlined items are hypertext links to WWW servers offering those resources. If we were using Mosaic to access this document, the underlined links would also show up as highlighted text and a few images would be on the page, too. In this instance, both of the hyperlinks point to hypertext documents, but they could have just as easily been links to an image file, an audio clip, an MPEG movie, a formatted text document, an ASCII document, a WAIS search utility, or a TELNET session to a library accessible from the Internet. As long as your local host can support the media type and the necessary helper applications are available*, a wide range of multimedia resources are at your fingertips.

If you're using a browser such as lynx, the arrow keys will jump from hyperlink to hyperlink with the right arrow key following a link and the left arrow key going back one page. Using Mosaic, move your mouse over the highlighted text and click. Just before clicking your mouse, notice that near the bottom of the window is a funny looking address.† This is the URL, or Uniform Resource Locator of the hyperlink. The URL address is made up of the protocol name, host name and file name for the resource. For instance, the URL

---

* Helper applications are external programs which may be needed to view or hear a WWW resource. Typical helper applications are **GhostView**, **mpeg_play**, and **xv** for Unix workstations; **UlawPlay**, **Simple Player** and **JPEGView** for Macintoshes. An effort is made to recommend tools available in the public domain. You can configure the type of helper applications your Mosaic browser will use, so quite a few additional options are available.

† Mosaic window references are for the X Windows client; the Macintosh and Microsoft Windows clients have a slightly different layout. You may need to turn on some options in the Mac or Windows clients to see things like the URL, but the information is available on each client.

http://sunsite.unc.edu/unchome.html

tells us this resource uses the Hypertext Transfer Protocol (HTTP) to connect to the host **sunsite.unc.edu** and display the Hypertext Markup Language (HTML) document **unchome.html**. As a new user of Mosaic, you don't necessarily need to know what all of that means, but I like to check out the hosts to which I'm about to connect. If it's in the middle of the afternoon and the network is pretty sluggish, I may decide to bypass a hyperlink that will take me to Europe or the West Coast and try it again the next morning or later in the evening.

After clicking on the hypertext link, the browser follows the URL, and after a few seconds, a connection will be made and you'll see the new resource. If you end up waiting a long time for the resource to appear, you can click on the spinning globe in the upper right corner of the Mosaic window. This will stop the HTTP transfer and will display any information it has already received.

As you traverse through the Internet, you will run into resources that are served by protocols other than HTTP -- Mosaic can access Gopher servers; it can perform WAIS searches; it can initiate TELNET or FTP sessions; it can run the **finger** command; and it can do many other things, too. Watching the URLs fly by is a great way to develop an understanding of how Mosaic works and just how powerful a tool it is.

Continuing your exploration of the Internet, you will soon discover that it is extremely easy to get lost in the vastness of the Web. Should you find yourself in such a situation, you have several options. At the bottom of the Mosaic window, there are a few buttons -- Back will move you back one document and Home will take you back to your home page. Under the Navigate menu at the top of the window, you can select Window History and choose a particular document you would like to return to. Another useful feature, also under the Navigate menu, is the Hotlist. As with Gopher, this allows you to mark a particular location for quick and easy access in the future. You can add the current document to your Hotlist, and you can view your Hotlist and select a site for a return visit. A final suggestion for navigating in the Web is to use the Open URL option under the File menu -- this allows you to enter any URL of your choosing. This is a very useful feature when you start collecting URLs from friends, news groups and other documents.

At this point, you are now ready to venture out on your own. The best method for learning how to use Mosaic is to simply dive in and start using it. There are so many features available, it would be impossible to cover all of them here. For more information on Mosaic or the World Wide Web, you really should look in the Web. There is quite a bit of documentation already out there -- many sites create their own documentation and make it available to the Internet; CERN is a wonderful source for introductory documentation on the Web and on creating your own server; NCSA also has documentation available. There are many other sites with useful information, too. Ironic as it may seem, I always recommend using Mosaic to find information about Mosaic and the Web. In the **For More Information** section, I have included some URLs to get your started.

Conclusion

I hope I have given you a head start on using and understanding Internet discovery and retrieval tools. The subject matter is vast, and this paper could go on for hundreds of pages and still not cover everything, but I'm confident you now have the knowledge and courage to face the Internet on your own. The Internet can be an overwhelming experience, given its size and the sheer amount of information that's out there; but it can also be a magical, exciting, fun and extremely useful experience. I encourage you to try it out, but be careful -- you just might get hooked!!

Acknowledgments

This paper would not have been possible without the experiences of all the Information Seekers who have gone before me. Simply put, I thank the Internet community and hope those who follow me find this small contribution a valuable resource.

References

[Krol92] Krol, Ed. *The Whole Internet User's Guide and Catalog.* O'Reilly & Associates, Inc. 1992.

[Grey93] Gray, Matthew. *World Wide Web Wanderer results.* MIT. http://www.mit.edu:8001/afs/sipb/user/mkgray/ht/web-growth.html.

For More Information

Read the USENET news groups:

comp.infosystems.gopher comp.infosystems.wais
comp.infosystems.www

Check out these books (and others like them):

Ed Krol, *The Whole Internet User's Guide and Catalog*, O'Reilly & Associates.
Tracy LaQuey Parker, *The Internet Companion*, Addison-Wesley.

Cruise the net:

http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/StartingPoints.html
Starting Points for Internet Exploration

http://www.rpi.edu/Internet/Guides/decemj/internet-tools.html
John December's Internet Tools Summary

http://info.cern.ch/hypertext/WWW/Talks/General.html
World-Wide Web: An Illustrated Seminar

http://sunsite.unc.edu/unchome.html
University of North Carolina at Chapel Hill home page

Author Information

Amy K. Kreiling worked for the National Center for Supercomputing Applications at the University of Illinois in Urbana-Champaign for six years prior to joining the Computer Services staff in the Department of Computer Science at the University of North Carolina at Chapel Hill. At NCSA, Amy was responsible for the administration of over 60 Silicon Graphics workstations. Along with providing supercomputer access to thousands of users around the globe, NCSA is known for its public domain software products such as NCSA Telnet, a large conglomeration of data analysis and visualization software, and the recent release of the immensely popular NCSA Mosaic, an Internet-based global hypermedia browser. Amy's current responsibilities include the administration of the CS Department's mail, news, and information systems, as well as the management of the Department's fleet of IBM RS/6000 workstations. Amy can be reached at kreiling@cs.unc.edu.

A Simple and Free System for Automated Network Backups

Karl A. Anderson
NASA Goddard Space Flight Center
Code 920.2
Greenbelt MD 20771
karl@ltpsun.gsfc.nasa.gov

Brian H. Kirouac
Hughes STX Corp.
NASA/GSFC code 920.2
Greenbelt MD 20771
bri@ltpsun.gsfc.nasa.gov

ABSTRACT

This paper presents a simple system of shell scripts for nightly unattended backup of multiple disk partitions on multiple machines, to one or more central tape drives. The system is based on standard UNIX utilities: cron, Bourne shell scripts, the BSD (r)dump program, and the BSD remote shell program. Standard high-capacity tape drives (e.g. 3480, 4mm, 8mm) are also used. The use of standard hardware and bundled software reduces budget constraints and eliminates paperwork hassles caused by the need to procure software licenses and support, and also ensures maximal portability. Security is provided by the use of a non-root backup userid, with appropriate measures to impede spoofing. Successful backups are logged, as well as a wide variety of error conditions. Operator intervention is minimal: the tapes must be changed and the log files examined daily.

The system described in this paper has been in use in the authors' organization for two years. In that time, dozens of files and several trashed filesystems have been restored. The system isn't perfect, but it is an example of how to create a workable system from freely available tools.

Why This Paper?

One of the first problems novice UNIX System Administrators must solve is backups. If users have the right to expect anything from someone who calls him or herself the System Administrator, it is that he or she keep the users' precious files backed up in case of disk failures, accidental deletions or other acts of God. In the bad old days, backups were done on 9-track or QIC tape drives of at most 150 Megabytes capacity. Disk partitions of any size filled up more than one tape, so an "operator" had to change tapes when prompted. Thankfully, helical-scan tape drives that store as much as 25 Gigabytes on a single tape are now available. At many sites, one of these drives can back up every desktop workstation without a tape change. That means backups can be automated. All the Sysadmin has to do is schedule the backup job under *cron*, then change one tape each day and everybody's happy.

Just how does one "set up the backup job"? UNIX has traditionally provided basic backup utilities, namely *pax, cpio, tar* and *dump*. With a little bit of effort, these traditional tools can be incorporated into a system of scripts to accomplish this mundane but essential task. Many such systems have been written; this paper describes one that is simple, reasonably robust, and relatively secure. Best of all, it's free.

Why go to the trouble? After all, a number of commercial backup packages are available that are easier to use, more secure, more reliable, etc. than the traditional tools. Vendor representatives can be found at this very conference, and are eager to talk to you. On the other hand, there are many reasons why a novice Sysadmin might be unwilling or unable to buy a commercial backup package. Limited or no budget is one obvious reason. But aside from the expense, there's the hassle factor. At many large government and academic installations, there is little central control over the purchase and allocation of hardware and software resources. This can make managing procurements a potentially infinite time sink. Purchasing client licenses for all workstations including new ones that arrive at random intervals, and renewing maintenance contracts on some annoyingly short period as required by the Sysadmin's management, can be non-trivial chores when a commercial backup system is used. Then there are problems like the unavailability of ports to the OS one happens to run at one's site, and the need to update the package when the OS is updated.

The system described herein uses utilities bundled with most vendors' versions of UNIX: *cron*, the Bourne Shell interpreter *sh*, *(r)dump*, and the BSD remote shell program *rsh* (or *remsh*). These are more or less compatible across a wide variety of vendor's UNIX platforms, and don't need to be purchased or installed separately. There are no kernel rebuilds, and no balky license managers to wrestle with. The bundled tools are documented in the man pages that most popular OSes come with. The tape drives don't have to be specially integrated with the software. Since any standard tape drive will do, and since the software tools are so widely available, the backup tapes are maximally portable.

The authors' system has been successfully installed on Sun workstations running SunOS 4.1.x and 5.3, SGI machines running IRIX 4.0.x and 5.x, and HPs running HP-UX 8.x and 9.x.

Overview

Each of the traditional UNIX backup utilities has at least one significant shortcoming - see Zwicky 1991 [1] for a detailed enumeration. The venerable BSD *dump* program was chosen as the best of a bad lot among the bundled or free utilities analyzed by Zwicky. Its chief drawback is that changes to files during the dump can cause unexpected results when the changed files are restored. *Dump* also imposes a filename length limit of 1021 characters. Otherwise, it passed all of Zwicky's tests.

The backup system itself requires minimal installation. It consists of three Bourne shell scripts: *backup_cluster.install*, the installation script; *mdump*, a script that runs *dump* for each filesystem on an individual workstation; and *backup_cluster*, a master script that runs on a dump host and invokes *mdump* on each client workstation in a list.

Dumps are done to high-capacity (e.g. 8mm) tape drives, with multiple filesystems being dumped to a single tape. There is no provision for multi-tape dumps; the total size of the filesystems dumped in each execution of the master script must be less than the capacity of a single tape. An Exabyte EXB-8505 compressing tape drive can hold 7.5 GB or more, so a few of these drives can cover a lot of machines.

Reporting and error handling are extensive. A wide variety of error conditions is reported to the operator by email and through log files. Successful dumps are reported as well as failures. The amount of data written to tape for each machine is also reported.

Security is moderate. *Backup_cluster and mdump* are executed by a non-root backup user, with a .rhosts file listing all the machines involved in the system. The security risks introduced by the use of .rhosts are well known, but are deemed acceptable at the author's site. The backup user has a starred password, so no one can log in as that user without first becoming superuser on a participating node. The backup user account is not distributed with NIS, but is placed in the local password file on each client. And the filesystem the backup user's home directory is on is exported only to the clients. The

backup user is a member of the group that owns the raw disk device files, which are made readable by group; this is the default for Suns, but must be modified on SGI and HP systems. Because the remote tape server *rmt* uses a reserved port, the *rdump* executable must be setuid root; again, this is the default for Suns, but not for SGIs or HPs. The required mode changes are done by the install script.

The system is distributed: *backup_cluster* can run on one or more master backup hosts, with the dumps from multiple groups of workstations going to separate tape hosts. For ease of maintenance, the backup user's home directory can reside on a single machine, and be automounted by all other machines that participate in the system. This is not required, however; in fact, for increased security, the backup user should be given a local home directory on every participating node, and *rdist* used to keep files synchronized.

Level 0 (full) dumps can be done using *mdump* as a stand-alone script. If a copy of *mdump* is placed in /etc, the dumps can be done at single-user level in an automated way. The boot scripts are modified to execute *mdump* when a flag file is present at boot. Full dumps are then scheduled by an *at* job that creates a flag file and reboots the workstation. The dumps are performed after the network interface has been configured and all the local filesystems have been fsck'ed, but before multi-user level is entered.

Incremental dumps are scheduled by *cron*, and require minimal operator attention: the tapes must be changed daily, and the log messages checked for errors. The dumps are done with the clients at multi-user level, so they should be scheduled for periods of low activity. Filesystems that are active around the clock, like mail spools, should be small so that the interval between mapping and writing to tape is short. The total time required for each job depends on the total disk space dumped, but the authors' experience is that a 5GB tape can be filled in about four hours.

The most common problem is failure to insert the day's tape. The next most common problem is that the incremental tape fills up. When that happens, level 0 dumps are scheduled on the machines that dumped the most data at incremental level. Also seen with depressing frequency are media errors due to defective 8mm tapes. The incidence of these is dramatically reduced when data-grade tapes are used. Some problems have been encountered when reading dump tapes made on tape hosts of different architectures, e.g. Sun and SGI. Even after byte-swapping and blocking issues are resolved, a tape made on a Sun may be unreadable on an SGI, and vice-versa. The solution is to restore over the network, from a tape drive on the same type of workstation the tape was made on.

Restoring is done with *dump*'s counterpart, *restore*. The correct dump file on the tape is found by looking in that tape's log file for the most recent dump written to the tape. The dump file for the desired filesystem is found by counting forward from the initial date stamp, and using *mt* to space forward on the tape. *Restore* has an interactive mode that makes restoring individual files fairly painless. For security reasons, *restore* should not be setuid-root; restores should be done by the superuser, after placing an entry for root on the client machine in the backup user's *.rhosts*. Remember to remove the entry for root from *.rhosts* when the restore is finished.

The system has been in use at the authors' site for about two years. At present, there are eight clusters of workstations defined, each dumping to a separate Exabyte EXB-8500 tape drive. Each cluster has an "operator" assigned, who takes the responsibility to change the tapes daily. The results of each job are emailed to the Sysadmins, who review them each day, and take steps to correct any problems that arise. Individual machines may go for a couple of days without a backup, but this is considered acceptable at the authors' site. During the time the system has been in operation, dozens of individual files and several trashed file systems have been restored.

Details

Setting the system up on a new machine consists of adding the backup user to */etc/passwd*, and creating the local home directory or entry in the automounter's home directory map. The */etc/dumpdates* file should be made writable by backup. On HP and SGI machines, the permissions on the raw disk device files must be changed so that backup's group can read them, and */usr/etc/dump* must be made setuid-root. These steps are performed by the install script, as appropriate for each architecture. Note: IRIX 5.1 has copies of *dump* in */sbin* and */usr/etc*, with a soft link in /etc pointing to the one in /sbin! *Mdump* puts */usr/etc* first in the search path, before /sbin, /etc, or '.'.

The master script, *backup_cluster*, executes under cron on a dump host. *Backup_cluster* accepts arguments for dump level, dump device in *host:device* syntax, and the name of the file containing the list of client machines. The list of clients may specify which fileystems are to be dumped on each client; the default is all local filesystems. Backup_cluster calls mdump with the specified dump level, device and filesystem list. Mdump in turn dumps each filesystem specified by backup_cluster or by default.

Separate instances of backup_cluster are executed for each cluster of clients, with the dumps for that cluster going to a single tape drive. Reports for each cluster are appended to a logfile named for the cluster. Each job's logging begins with a date and time stamp; results for each client begin with the name of the client and the time mdump began on that client, followed by a list of filesystems dumped for that client, with the values of the dump parameters shown. The total amount of data dumped for each client follows the list of filesystems. Remote shell, dump and tape error messages also appear at the point they occurred. Each job's log output is copied to stdout as well as appended to the logfile, so that when backup_cluster runs under cron, anyone who is aliased to the backup user receives the report by email.

Backup_cluster redirects remote shell and *mdump* error messages to a <client>.out file. *Dump* itself writes voluminous status messages to stderr, which are also captured in <client>.out. After *mdump* finishes on each client, *backup_cluster* scans <client>.out for error messages of interest, logging any it finds to the logfile as well as to stdout. *Backup_cluster* also extracts and logs the amount of data dumped by each client from the <client>.out file. *Backup_cluster* and *mdump* both perform checks of tape drive status at the beginning of each job. *Mdump* also checks tape status after each filesystem is dumped. Mdump also reports the successful completion of each dump on stdout; these messages are logged by *backup_cluster* as dump errors are.

As mentioned above, *mdump* was designed for standalone use. It accepts arguments for dump level, dump device in user@host:device syntax and the list of filesystems to be dumped. Mdump also accepts a switch to take the tape offline when it is finished, as a safeguard against being overwritten accidentally.

Examples

The names of machines in the examples that follow are fictitious. Here's a sample crontab for user "backup" on dump host "cedar":

```
root@cedar% crontab -l backup
# cedar clients
0 1 * * 2-6 ./backup_cluster -l 1 -d cedar:/dev/nrst9 -r clients.cedar
# circus clients
0 1 * * 2-6 ./backup_cluster -l 1 -d circus:/dev/nrtapensv -r clients.circus
```

Every week night at 1:00 AM in this example, two instances of *backup_cluster* run on cedar. The first does a level 9 dump of all the clients in "clients.cedar" to cedar's own 5GB 8mm tape drive. The second does the same for the machines in "clients.circus", to circus's 8mm drive. Cedar is a Sun workstation, and its dump clients are Suns and HP workstations; circus and its clients are SGI workstations. The client file consists of lines containing client nodenames and an optional list of filesystems. Comments are also allowed. If a list of filesystems follows the node name, *mdump* dumps only those filesystems. Otherwise, all local filesystems listed in the client's */etc/fstab* (or its equivalent) are dumped.

Here is cedar.clients, showing the format of the client file:

```
root@cedar% cat ~backup/cedar.clients
# Alice's HP 735.
redqueen                        # dump all filesystems of type hfs
# Jack's HP 710.
beanstalk                       # ditto

# prairie project Suns.
prairie                         # dump all filesystems of type 4.2
poa                             # ditto
bluestem / /usr /home /data     # don't dump scratch disk
```

Here is some sample log output for cedar's clients, illustrating the various errors logged as well as successful dumps:

```
root@cedar% cat backup_cluster.cedar.log
...[lines deleted]...
Tue Nov 23 01:00:04 EST 1993
/dev/nrst9: no tape loaded or drive offline
Wed Nov 24 01:00:02 EST 1993
redqueen: remshd: Login incorrect
beanstalk: Wed Nov 24 01:01:53 EST 1993
 dump 1ubdsf 112 54000 12000 cedar:/dev/nrst9 /
 197422 kilobytes written to tape
prairie: Wed Nov 24 01:14:24 EST 1993
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /home
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /usr
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /data
 1164477 kilobytes written to tape
poa: Wed Nov 24 02:11:22 EST 1993
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /home
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /usr
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /var
 2191738 kilobytes written to tape
bluestem: Wed Nov 24 03:41:01 EST 1993
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /usr
 dump 1ubdsf 112 54000 12000 backup@cedar:/dev/nrst9 /home
 dump of /data aborted
Exabyte EXB-8500 8mm tape drive: sense key(0x13)= EOT residual= 0 retries= 0 file no= 66 block no= 0
 727543 kilobytes written to tape
all hosts: 4281180 kilobytes written to tape on Wed Nov 24 10:08:44 EST 1993
```

The problem on Tuesday morning is simple: the "operator" forgot to put in the tape. That was easy to fix. On Wednesday morning, we see that redqueen didn't allow cedar to run a remote shell. On investigation, it turned out that redqueen's automounter wasn't running. Once that was fixed, the next night's job got redqueen along with the rest. Wednesday's job also filled up the tape. The solution to that was to do a full dump of poa.

Tape host circus is an SGI. Sample log output for circus's clients shows how tape status messages differ between Sun and SGI tape hosts:

```
root@cedar% cat backup_cluster.circus.log
...[lines deleted]...
ringling: Sat Dec 4 01:00:03 EST 1993
 dump 1ubdsf 112 54000 12000 backup@circus:/dev/nrtapensv /
 dump 1ubdsf 112 54000 12000 backup@circus:/dev/rmt/nrtapensv /usr
 dump 1ubdsf 112 54000 12000 backup@circus:/dev/rmt/nrtapensv /usr2
 dump of /usr3 aborted
Controller: SCSI Device: EXABYTE: EXB-850085BANXR40428 Status: 0x20261 Drive type: 8mm cartridge Media :
READY, writable, at EOT, block 40025
...[lines deleted]...
```

This example also shows the importance of performing full dumps frequently.

Conclusion

The system described in this paper isn't perfect, but it is an example of how to create a workable system from freely available tools. If money is available and purchasing overhead is acceptable, a commercial package may provide more security and reliability, as well as features like a GUI for users and administrators. If, however, budget is limited, security and reliability requirements are moderate and a command-line interface is acceptable, a system such as the one described may meet the needs of even the experienced Sysadmin. If nothing else, it was a fun programming exercise.

Availability

A shar file containing *backup_cluster*, *mdump* and *backup_cluster.install* can be obtained by anonymous ftp to ltpsun.gsfc.nasa.gov, in */pub/backup_cluster.shar*. It is also available by *gopher* from the same address. The authors make no warranty regarding the use of any of the scripts.

References

1. Zwicky, Elizabeth D., 1991, "Torture-testing backup and archive programs: things you ought to know but probably would rather not." Proceedings of the USENIX Fifth Large Installation System Administrator's Conference, San Diego, CA, pp. 181-189.

Building an integrated and enterprise-specific configuration management solution

Jan Gottschick, Malte Timmermann
Fraunhofer Institute for Software Engineering and Systems Engineering
Kurstr. 33, D-10117 Berlin, Germany
E-Mail: Jan.Gottschick@isst.fhg.de, Malte.Timmermann@isst.fhg.de

Abstract:

The ICOMA[1] system is intended as a flexible and open platform for configuration management. ICOMA should be able to manage your own and outside applications, no matter what size your network is or how your organization is structured. It is adaptable to the administrators personal requirements. The volume of data to be given by the administrators should be minimized.

ICOMA allows you to write your own managed objects (MO), which configure your applications, with the support of development tools. It is possible to combine existing managed objects to form new compound managed objects. ICOMA supports a repository which scheme models your enterprise and whose contents are used to speed up the ICOMA system and to make it more reliable. The communication between managed objects and the ICOMA tools is handled by the configuration bus which supports wide distributed environments. The MO-Editor of the ICOMA system is a generic management tool to configure most of the managed objects. It supports configurable user dialogs. That allows the strict division of the user interface from the functionality of the managed objects. The ICOMA system can integrate and can be integrated into existing and emerging standards like SNMPv2[2] using gateways.

Introduction

In recent years more and more end users got a computerized workplace. These computers are mostly standalone systems. The management of these systems is carried out by the users themselves or by a local administrator, who does this job during his normal work. Due to the spread of this technology the users now no longer want to work isolated from their colleagues. They want to share data as well as to communicate with their computers. That means the systems have to be connected to a network, and the systems or the applications have to cooperate. The isolated solutions have to be integrated.

From the viewpoint of the professional administrator the world has changed from centralized systems to distributed, cooperating systems. This change is most often a slow process. Some user groups installed small systems of their own which worked autonomously and were administrated by the users themselves. Now the system managers have to integrate these distributed systems and they find a situation where many different kinds of machines exist. Furthermore, the number of installed systems and required applications is growing very rapidly which makes the management of these applications increasingly complex. To manage these difficulties a growing number of system administrators is needed which is not only proportional to the number of machines but also to the integration and coordination problems.

To manage a complete enterprise environment of computer systems you can divide it into domains. One administration center is responsible for each domain. To coordinate the work and to solve common problems you need a central administration center parallel to the local administration centers. The local administration centers are partly autonomous in their work. They share services and exchange configuration data with the central administration center. This cooperation guaranties that the input of large volumes of data is decentralized and carried out at the locations of the data. With simple and convenient tools is it possible that this work can be done by the normal staff members and not the administrators.

1. Integrated Configuration Management
2. Simple Management Network Protocol [SMNP]

The scenario in Figure 1 has a local and a central domain. In this case the central domain keeps the data for the global services like E-Mail, X.500 or the phone list. The local domain manages parts of its own data. Services like NIS[1] distribute the information by themselves, while the local domains exchange data with the central administration center. The data can also be held completely centrally. It depends on your priorities. In both solutions the data of the user application should be managed at the local administration center. That doesn't depend on where the functionality of the configuration process is realized. The managed objects used in the example in Figure 1 have different implementations. There are a lot of different techniques for implementing a configuration procedure. Some techniques directly manipulate an agent, for example SNMP, CMIP[2], OSF[3]/OMG[4]-CORBA[5] and shell scripts. Other techniques configure the managed objects indirectly. They change their configuration data, for example X.500 [X.500], OSF-Cell Directory Service and Global Directory Service [DCE], Network Information Service (NIS+), domain name service (DNS, hesiod) or configuration files. The next point is that two local domains have to manage different managed objects. They have different configuration processes, too. They could differ, for example, in their default values for the managed objects. And finally, the various domains, and possibly the administrators, want their own specific view on their data and to reduce the input of data to a minimum. In the example (Figure 1) our administrator needs to input only the following data: first name, last name, phone number, room, user name, group name, home area, quota, mail (y/n) and directory (y/n). All locally and globally managed objects have to be configured from this data.

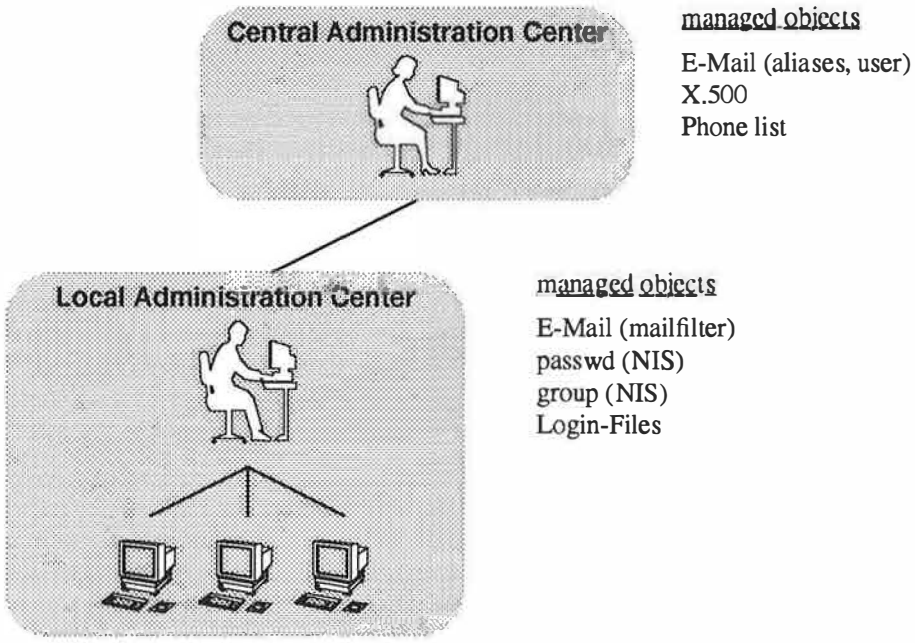


Figure 1: Administration Domains with their responsibilities

At the Fraunhofer Institute for Software Engineering and Systems Engineering (ISST) a configuration system is under development. The main goal was to have a flexible, modularized, scalable and distributed platform for configuration management. The administrator should have a common user interface for his configuration task independent of the kind of configuration process. He can also adapt his input dialog to his own requirements. The managed objects should be implemented as components, which could be combined to new managed objects, for example to configurate all managed objects from the example under the control

1. Network Information System
2. Common Management Information Protocol [CMIP]
3. Open Software Foundation
4. Object Management Group
5. Common Object Request Broker Architecture [CORBA]

of one new composite managed object. It should thus be quite simple to implement a new managed object, for example for a new application. The solution should be used in small sites and also in large sites with up to several tens of thousands of computers. In particular that means you can manage a very large network of computers.

The ICOMA system is running on the operating system Unix™ and managed objects will be available on PCs running Windows/NT™ from Microsoft Corp. The system routines are implemented in C++ with a small part in C. The shell scripts are implemented in "perl". The configuration bus is based on Orbix™, the CORBA implementation from IONA Technologies Ltd. and the graphical user interface is based on the Object Interface Library™ from ParcPlace Systems, Inc.

Overview of the architecture

Figure 2 gives you an overview of the ICOMA system.

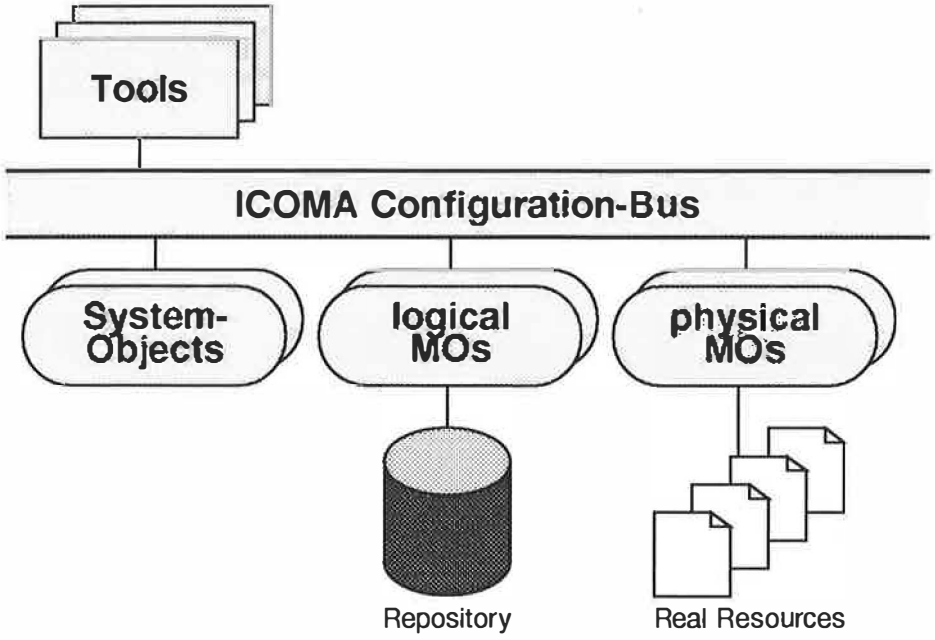


Figure 2: Architecture of ICOMA

The core of ICOMA is the configuration bus. It is based on an object request broker as defined by the OMG CORBA specification. The repository stores all enterprise specific data. The managed objects describe abstract manageable resources and are a uniform way to manage them. The ICOMA system views them as lists of entries. These lists are represented by tables. Each row represents an entry with its attributes. The real resources are the specific configuration of the applications. We have specified a Definition for the managed objects using IDL. It includes methods to add, change and remove entries of managed objects including error handling and an update request, to signal the change of a managed object by another administrator. It is a simple but generic interface to the managed objects. The main function of the configuration bus is to interconnect all managed objects and the tools with an interoperable communication mechanism. The CORBA technology was selected because it is very simple to create client and server applications especially using a C++ API. Furthermore, the object request broker allows the distribution of the managed objects over a local network. The configuration bus supports the integration of new components into the ICOMA system without recompilation of tools or other managed objects, as long as the interface definitions do not change. In the future it is planned to extend the functionality of the configuration bus, mainly to add concepts like access control and transactions.

Repository

The repository is the database for the ICOMA system. As shown in Figure 3 it is subdivided into subrepositories. Every managed object can have a subrepository, which stores the configuration data of a managed object persistently. The entire repository stores the data redundantly. That was done for performance reasons and to add new components easily without changing an enterprise-wide information model. As mentioned above the managed objects can be modelled using a table. These tables are implemented by text files. In the future the repository will be implemented using ISAM-files and with SQL-based relational databases. The consistency of the repository is an unsolved problem. A nonstandard transaction mechanism for a synchronized and consistent update is needed and also consistency conditions for the repository. Today a simple mechanism is used which manages unique keys for values of the managed objects which belong together.

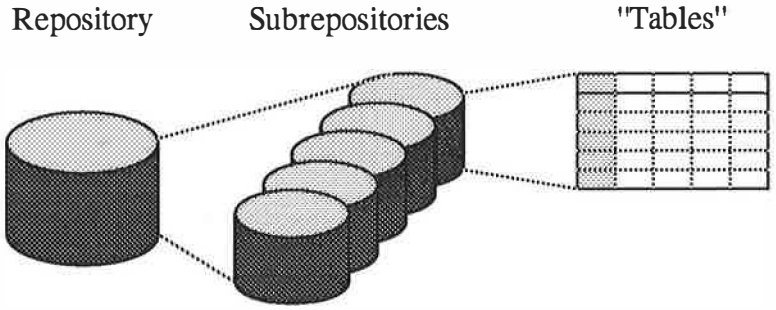


Figure 3: Repository substructures

Managed Objects

A managed object is the base unit in ICOMA. They can be distributed over several systems in a network. They consist of lists of entries which are the atomar units which can be manipulated. It was decided not to use the attributes of an entry as an atomar unit, because one can get inconsistent states if the attributes are changed independently. For example, there are two entries (Jan,Gottschick) and (George,Gottschick) and the first entry shall be changed to (George,Michael). If the first attribute is cganged first the second entry is temporarily the result. If these two attributes are used inside the managed object as a key for another kind of managed object, the result is undefined. On the other hand the runtime of the configuration process can be optimized by manipulating the whole entry at one time.

The configuration bus allows the integration of different kinds of configuration methods. The most important goal was to configure existing applications without changing them. Applications are configured for example by editing ASCII[1]-based configuration files or running a command line interface. Both techniques allow the use of shell scripts to configure such applications. An adaptor to use shell scripts was implemented, which configures the real resources. Several managed objects are implemented by scripts in the language "perl". The adaptor handles the subrepository and calls the script. The attributes are passed through a pipe so that there are no special limitations to these attributes. At the moment the language perl is used to implement the scripts but other languages like tcl/tk can also be used. It is also possible to use programs written, for example, in C, if the attribute passing mechanism is implemented, which is really simple. On the other hand the use of other managed objects inside the shell scripts is possible. There is a small program which can be used to call other managed objects via the configuration bus. The attributes are passed through a pipe into the scripts. Furthermore, there is a C++ Library for direct use of the configuration bus.

Two tools were developed to simplify the job of writing scripts for new managed objects. The first tool is a compiler for the "Integrated Configuration Language" (ICOLA). The language ICOLA allows it to describe the implementation of a managed object in a common and easy way for all different kinds of configuration method. Each managed object has a signature and an implementation part. The signature describes the inter-

1. American standard code for information interchange

face of the managed object. It defines mainly the attributes. The signature is comparable to the MIB[1] of standards like SNMP but more simple. The signature is to be replaced at a later stage by the SNMP/MIB or the DMTF/DMI description. The signature is used by other managed objects and the tools. It defines on one hand the structure of the subrepository and on the other hand the sequence of attributes which are passed over the configuration bus with the method calls add, change and remove. It further guaranties that the different managed objects and tools use the same managed object definition, especially the same sequence of attributes.

The implementation part of the ICOLA-classes depends on the type of the configuration method. For SNMP, CMIP, NIS+, DNS,DCE/CDS & GDS and X.500 a description of the mapping of the attributes is necessary. To describe a managed object,, for example a shell script written in "perl", only the special core functions for the configuration process of your managed object must be implemented. The code for integrating these functions into the configuration process is generated by the ICOLA-compiler. Additionally, you can use utility functions which are also generated by the compiler. The goal is to minimize the effort required to implement a new managed object. Figure 4 shows an example of an ICOLA class. The result of thiis example will be a shell script which work as a managed object in connection with the adapter for shell scripts.

```
SIGNATURE NIS_passwd;
PARAMETER organization_name;
ATTRIBUTE user, UID, GID, firstname, lastname, home;
ENDSIGNATURE NIS_passwd.

CLASS NIS_passwd;
LANGUAGE perl;
ENVIRONMENT NISTBLDIR;
METHOD add; {
    # add entry in passwd
    open (nis_fh,">>%NISTBLDIR%/passwd")||
      &error("Cannot append to file passwd (NIS) .");
    print nis_fh " %user%::%UID%:%GID%:%firstname% \
      %lastname%:/home/%home%/%user%:/bin/tcsh";
    close(nis_fh);
    &report("The user %user% was added to the NIS passwd");
}
METHOD remove; {
    ...
}
METHOD change; {

    ...
}
ENDCLASS NIS_passwd.
```

Figure 4: Signature and Implementation of a managed object

The first four lines describes the signature part of the managed object. The key words SIGNATURE and ENDSIGNATURE mark the beginning and the end of the description. The key word PARAMETER is followed by a variable for the definition of an organization structure. The third line describes the structure of the repository for this class. Especially the order of the attributes is defined here.

The lines from the key word CLASS to the key word ENDCLASS describe the implementation part of the managed object. The key word LANGUAGE is followed by the destination language to be used. The next line describes a variable which is added to the environment and can be used by every subsequent script. The next lines define the implementation for the different method calls add, remove and change. All lines of code

1. Management Information Base

for one method call start with the key word METHOD and must be grouped by braces. That code parts are written in the target language of managed object.

The ICOMA development tool facilitates development of ICOLA-classes. The developer has a comfortable user interface to generate the classes. The tool has browsers to manage the known classes. Both tools are still under development.

The managed objects can be classified into logical and physical managed objects. The physical managed objects only configure real resources. They don't configure other managed objects. The logical managed objects only configure other managed objects. They don't configure real resources. This distinction will faciliate the introduction of some new concepts. The logical managed object models the repository of an enterprise. The logical managed objects also can be used to combine existing managed objects to a new compound managed object. Logical managed objects can be realized which implement an enterprise or administrator specific configuration task or specific options. That allows the implementation of logical managed objects which need only a minimum of input data for the configuration process. So there is a toolbox with reusable managed objects which can be flexibly combined and supplemented by new managed objects. The managed objects can be developed independently, and used by other developers, because the managed objects have a well defined interface. The managed objects for the real resources should be developed by specialists for the application which have to be configured. Thus only the specialists for an application have to know how this application is really configured.

Figure 5 shows an example of how the managed objects can be combined and used. The lower three objects are physical managed objects. They are used by a compound managed object, which implemented the logical user management. This compound managed object is used by the enterprise specific managed objects, which implement options which are specific to the departments of the institute at different locations. They get their input, for example, from the MO-Editor.

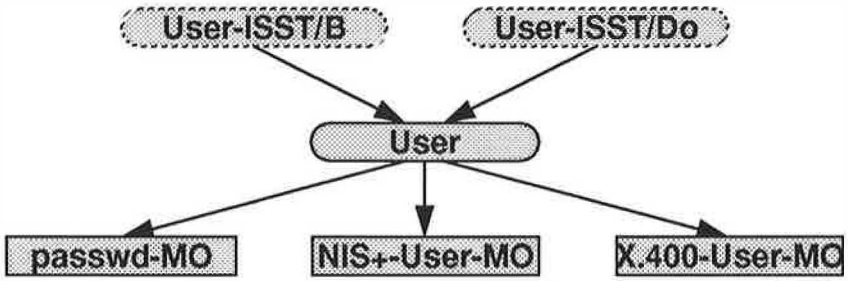


Figure 5: Combining managed objects

In a distributed environment the configuration management needs a security environment. The managed objects are distributed over several systems in the local and perhaps wide area network and several administrators with different competences manage the various applications. The configuration bus supports filters, which assign an access control list to each managed object. The access control list contains a list of the permitted administrators for this managed object.

MO-Editor

An important goal of the ICOMA system is to have common and user-friendly tools. The most important tool for the administrator is the configuration tool, which is called MO-Editor inside the ICOMA system. This tool is generic in the sense that it can be configurated to support the various managed objects. The graphical user interface is easy to use and the managed objects are represented in a common way. It can be used for all managed objects which are available via the configuration bus. The representation of the input dialog for the attributes of an entry of a managed object can be changed to the enterprise and administrator specific requirements. Furthermore, the MO-Editor supports the concept of specific views for an administrator.

The most important experience at administration centers is that you need a common user interface for all configuration tasks. A handbook with informal descriptions of how to configurate each system is not practical. That means, it is necessary to have an automatic script for doing the configuration tasks.

The MO-Editor offers the same kind of dialog for all managed objects. The layout of the dialog window is automatic generated at popup time. The attributes are automatically positioned in the dialog window using simple rules. The administrator has the possibility to configurate the representation for every managed object. Standard types are used for the attributes which map to a standard motif widget. The following types are supported by the MO-Editor: string, integer, real, boolean, date, open and fixed popup list of strings. Furthermore you have the possibility to change the layout of the dialog using the following elements: groups of attributes or groups, framebox, background pictures, overlay layers and fonts. It is possible to define default values or to hide attributes. The labels of the input fields can also be changed. That allows the administrator to customize his interface to a wide range of managed objects. Figure 6 shows an example for the description of the representation of an input dialog.

```
REPRESENTATION User SIGNATURE User:
ICON = "User.xpm";
{ LAYERS [ FONT = "-adobe-courier-*-*-*-*-*-*-*-*-*-*-*-*" ] {
    layer_1 (8,415,75,35) { PICTURE ("Background1.xpm") {
        User = STRING { INFO = "Username";};
        ...
        Telephone = INT { INFO = "Phone"; DEFAULT = "200"; };
    }; };
    layer_2 (105,415,75,35) { PICTURE ("Background2.xpm") {
        LABEL ( "Options" ) ;
        Mail = BOOL { INFO = "Mail"; DEFAULT = "Y"; };
        Fax = BOOL { INFO = "FAX"; DEFAULT = "Y"; };
        Directory = BOOL { INFO = "White Pages"; DEFAULT = "N"; };
    ...
}.
```

Figure 6: Example for a representation file

The first line contains the name of the representation and the name of the signature file. The next line gives the name of the icon which is used to represent every entry of a repository. The following block marked by braces contains the layout description. The first key word LAYER defines a layer layout and the following key word FONT defines the font for all text on this layer. The next line - starting with layer_1 - is the beginning of the description of the first layer. This line also contains the position and size of this layer as well as the name of the background pixmap. The next block marked by braces contains the different attribute descriptions onto the layer. The first word is the name of the attribute, for example User. The word after the equals sign is a key word describing the type of the attribute. The inner block also marked by braces contains the single definitions for the attribute layout. For example the key word INFO is followed by a text which is written in front of the input field or the key word DEFAULT which gives a default value for the attribute.

The main area not supported is configuration tasks which need a special graphical representation. One example is a HUB, where you can see the real image of its front panel with all lamps and switches which can be configurated by pressing the switches.

The configuration file for customizing the representation is rather simple. It is some simple sort of user interface language specialized to our needs. The configuration file also defines the relationship to the ICOLA signature. At the moment there is no graphical interface builder for the MO-Editor dialogs, but the realization is planned for the future. Figure 7 shows a screendump of the MO-Editor.

The concept of the MO-Editor and the logical managed objects allows you to differentiate strictly between the user interface and the functionality of your configuration process. You can develop independent managed objects for all your real resources and adapt them to your enterprise or administrator needs. The config-

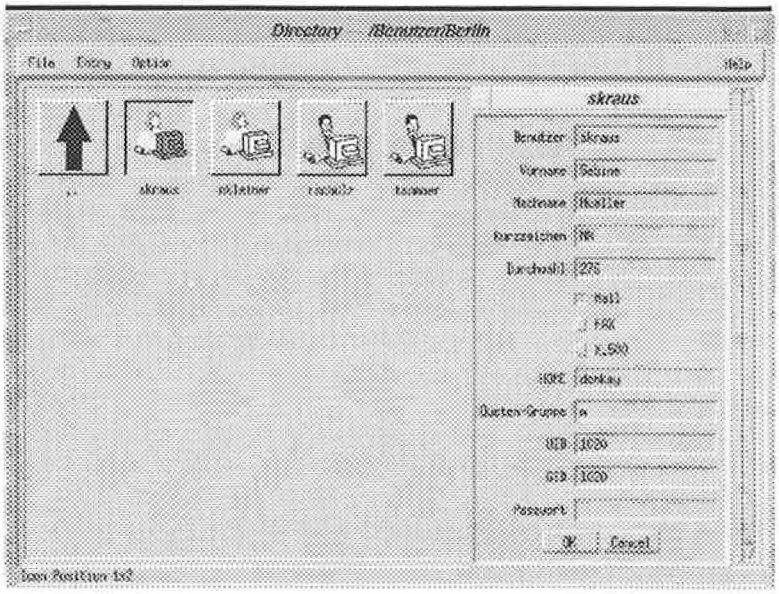


Figure 7: ICOMA MO-Editor

uration file for the representation of a managed object in the MO-Editor allows a lot of options to be set without changing the signature or implementation of a managed object.

The second important concept of the MO-Editor is the administration view. The managed objects are hierarchically structured as a tree in the browser of the MO-Editor. You can browse through this tree to find the entries of a managed object which the administrator can manage. The tree can be supplemented by folders to sort the managed objects. The administration view can be configured by the administrator to his specific needs. The administration view defines the mapping between the entries in the tree and the instances of the managed objects. A special case of the administration view is the topmost level of the tree. It defines different roles under which an administrator can act. Figure 8 shows a example.

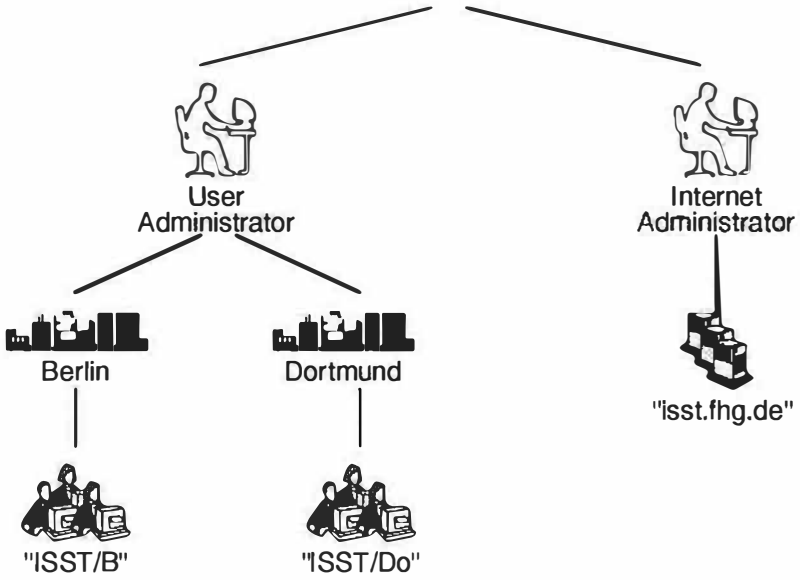


Figure 8: Administration view

This administration view defines the two roles “User Administrator” and “Internet Administrator”. The “User Administrator“ manages Berlin users in one folder and Dortmund users in the other one. The “Internet

Administrator" manages all computers of the ISST. There are two managed objects "Users" and "hosts". The managed object "Users" has two instantiations with different options set, for example the attribute location is fixed to Berlin or to Dortmund.

Gateways

We can implement gateways to adapt the configuration bus to foreign configuration systems which use a direct configuration technique. The aim is for the ICOMA system to use existing agents and for the SNMP manager to use ICOMA managed objects. On the one hand the gateway needs a generic managed object, which translates the method calls via the configuration bus into SNMP requests and on the other hand it needs a SNMP agent, which forwards requests for variable values, mapped to attributes of an ICOMA managed object, to the configuration bus.

Figure 9 shows an example for such a SNMP gateway. The generic proxy managed object needs a configuration file with the mapping of the attributes to the MIB variables or tables of SNMPv2. It doesn't handle traps. Its counterpart, the generic proxy agent, has to handle SNMP requests. The most important role of the proxy agent is the combination of sets of requests to one managed object into one method call for the configuration bus. The mapping of the SNMP oids to the managed objects and their attributes is adequate to the proxy managed object.

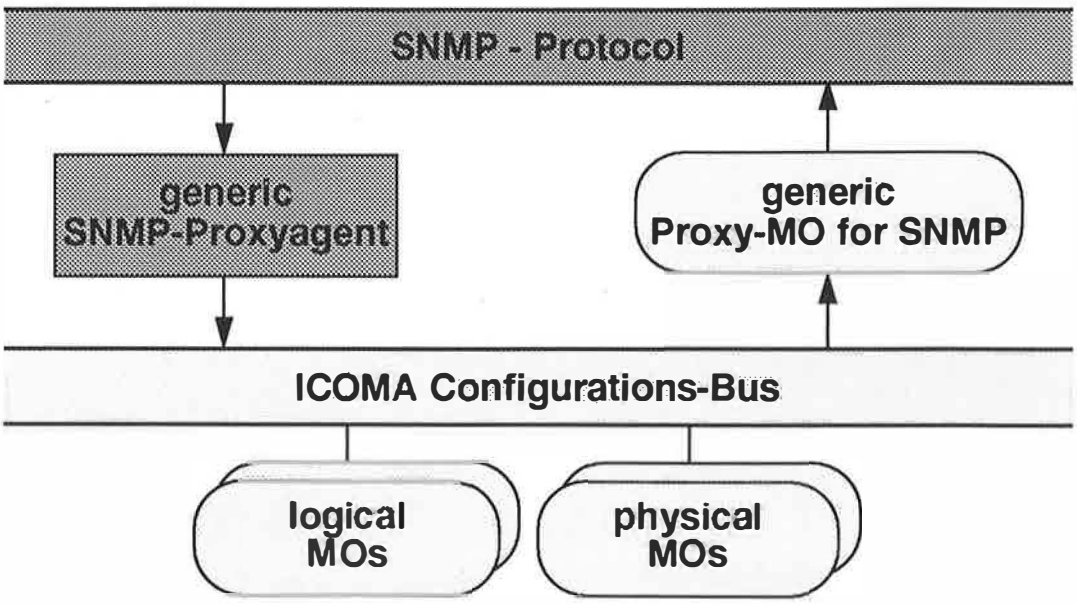


Figure 9: SNMP gateway to the ICOMA configuration bus

This technique can also be used to bind two different configuration busses. That could be necessary if autonomous domains are using different products or to isolate the object request broker for security proposes. A new protocol is needed, for example an RPC-protocol, which maps the functionality of the configuration bus. This protocol can be added by the management functionality of the configuration bus.

The gateway to the common configuration systems, which use the indirect technique, is much simpler. You need only a generic proxy managed object. That can be implemented for example for NIS+, DNS, OSF/CDS & GDS and X.500. You only need the mapping from the attributes name to the names of the foreign configuration system.

Logging of the configuration processes

A configuration process can be very complicated and need a long runtime. Further configuration processes can be initiated by several administrators and perhaps automatically, for example every night. A configuration process can also be part of an installation process. In such an environment it is important to watch the progress of the configuration processes. The ICOMA managed objects send several messages to document the progress. At the moment the messages are collected by the syslogd of the unix system. Later it should be possible to send these messages to our own "broadcast abonnement message service" (BAMS). BAMS is a general service to automatically distribute messages over a local or wide area network to the applications which are interested in messages of this class and from this source. The format of the messages is well defined and includes especially the time, source, class, severity and information of the message. Every application can subscribe to a class of messages at a BAMS-Server. On the other hand applications can send messages to a BAMS-Server and it will distribute the message to all interested parties. BAMS is comparable to SUNs tooltalk, which has nearly the same mechanism for local applications, and to HP event message service on HP/UX. BAMS uses the remote operations of "isode" as transport mechanism. ICOMA uses this service because it is simple to use and will also work in a wide area environment.

These collected messages should be filtered and controlled by the administrator who initiates the configuration process, and by the operator, who controls the global state of the systems. The source included in the message has an unique identifier, which enable tools like our "Job Monitor" to relate messages to one compound process. The job monitor is a tool to keep track of the progress of a job, for example a configuration process. A job is a list of messages with the same context. The job monitor shows you at the first level the summary of a job including its state and the last message. For example you can see at once whether a job is still running, finished or had an error. At a second level you can view all messages belonging to this job. The job monitor is still under development.

Experiences

The ICOMA system has been being used by two institutes of the Fraunhofer Society for over two years now. The old version currently used has shown that the system is simple to use and reliable. The administrators concerned needed only a very short time to learn it, although the old ICOMA system has no graphical user interface. The administrator had to edit text files and to start the configuration process with the make utility. This primitive but common user interface has shown that the number of errors due to wrong configuration had rapidly decreased. To add new managed objects to the existing configuration process was only a small problem. The administrators recognized the change only when the format of the files they have to edit changes. The greater problem was errors in the configuration scripts. Until the scripts were free of errors this part of the configuration process was not reliable and a specialist who knew the configured application well was needed to write the script. The good thing was that the other parts of the configuration process were normally not disrupted because the different processes are normally independent. The scripts also proved to be well reusable.

Another positive effect was that there was no problem in temporary replacement of an administrator. ICOMA has a simple and common user interface so every administrator normally knows how to use it. In the past we used a handbook to describe all necessary tasks but the problem was that the documentation was not always well understood.

The modularity of the ICOMA system had another good aspect. In the past we also had automatic scripts. But they were not very well modularized. So, it was normally not possible for two or more people to work with the system. Now you can distribute the development of the configuration process easily to different people. Normally the administrators have to develop the configuration scripts responsible for an application. He needs a little more time for installation and adaptation of the software, because he has to write the scripts. But the other administrators need no information on how to configure this application because it should be included in the existing system. They need only to know the meaning of the new attributes which have been added to the end user interface. Furthermore, you have to remember that configuration scripts are only needed when the software has to be configurated continuously, for example for user management.

The new version will soon be used by the two institutes. We expect that input errors thanks to the new user interface will decrease again. The configuration process will also be improved. Now we can configurate not only applications at the central management station but also at every client station. In addition we want to use the ICOMA system together with the second location of our institute in a field test. We want to share the tasks between our staff at both sites.

The problems left to solve

The most important problem to solve is to simplify the development of new managed objects. We have to implement our ICOLA compiler and to collect user experiences. We have to better understand and implement the parametrization of the managed objects. In this context we are examining the usability of inheritance for our ICOLA language.

The second problem area are is consistence of the ICOMA repository. We have to implement mechanisms for synchronization and compensation. The compensation will be needed to undo the finished actions if an error occurs. The compensation is part of the concept for nonstandard transactions. Another unsolved problem is the update of a subrepository to a new version, with a new scheme. We need a general mechanism for this update.

Conclusion

The ICOMA system is a base platform for the integration of configuration management. The reusable managed objects allow the development of the system in a modular fashion. The concept of logical managed objects permit the flexible adaptation of the system to the enterprise and administrators needs. You can scale the ICOMA system to different user needs. On one hand the communication facilities of the configuration bus, including the possibility to connect two configuration busses via a gateway, enable us to manage not only local networks but also wide distributed systems. On the other hand it is possible to use different technologies to implement the repository including the synchronization and transaction concepts in the future. The configuration bus with its common protocol and integration possibilities can also integrate other configuration systems. Furthermore the tools of the ICOMA system are generic, for example the MO-Editor, so they can be used with all managed objects.

The ICOMA system is adaptable to the various needs of the administrators through the administration view. They can build their customized management environment. The use of one's own managed objects is the key to creating highly integrated configuration management for the system and enterprise specific applications. The security functions, together with any compound managed objects implemented, allow the distribution of the administration task to multiple administrators. Furthermore, the development of additions to the enterprise specific configuration processes can be distributed to multiple developers.

The straightforward and common user interface simplifies the tasks of configuration management. The configuration process itself can be monitored by the generated logs. That is a good platform to centralize the configuration management in a support center and to free the end user from the system management task. That will reduce the costs for the support of computer systems in the future.

Literature

[CMIP] Information Processing Systems - Open Systems Interconnection - Common Management Information Protocol Specification (CMIP); ISO 9596, International Organization for Standardization (ISO), 1990.

[CORBA] The Common Object Request Broker: Architecture and Specification. Object Management Group, Document Number 91.12.1, 1991.

[X.500] Data Communication Networks Directory; Recommendations X.500 - X.521, CCITT, 1988.

[DCE] Introduction to OSF DCE. Open Software Foundation, Prentice Hall, 1992.

[DMI] Desktop Management Interface Specification, DRAFT 4.2, Desktop Management Task Force, 1994.

"Make" as a System Administration Tool.

Bjorn Satdeva

/sys/admin, inc.

ABSTRACT

Some tasks which are part of providing automatic maintenance for a UNIX system can be thought of in terms of dependencies between files. The UNIX *make* program is very well suited to assist in such tasks. This paper outlines a number of UNIX system administration solutions, which involves *make* in some form.

1. Introduction

The UNIX *make(1)* program is traditionally thought about as a programmer's development tool. By design, its purpose is to build file(s) from other files, through a set of specified actions (for example, building an executable by compiling and linking a number of source files). The *make* program decides when a compile (or set of compiles) is necessary, based on the dependencies between and among source and object files.

This basic functionality also lets *make* be a useful tool to perform certain routine UNIX system administration tasks. One classic example of this usage is the Makefile used to maintain NIS maps on a NIS server. In the case of NIS, as well as many other similar cases, the *make* program can be executed on an hourly basis from *cron*, without wasting much CPU time, because *make*, with its dependency rules, is able to decide whether any real work need to take place.

This paper will show several other practical application where *make* and file dependency rules are part of the solution.

2. Saving important system files

A common problem encountered in UNIX administration is diagnosing the problem after one or more system files such as */etc/passwd*, */etc/group* or */etc/hosts* has been modified and the system starts to display some kind of wrong behavior. At many sites, this is handled by logging everybody changes a file, using a version control system, such as SCCS (Source Code Control System) or RCS (Revision Control System). If this is done for each change, it will leave a clear audit trail of when, why, and by whom each change has been performed. However, system administrators are only human, and even with the best intentions, they might not always remember to check in the changes.

Therefore, if a system starts misbehaving, it might be necessary to go back through the backups for that particular part of the system, which can both be time consuming and inconvenient if the system does not function correctly. It is therefore desirable to automate tracking of all changes made to the systems configuration, while preserving the previous configuration in an manner which is reasonably easy to access.

The makefile shown in Appendix A is one such possible solution to this problem. It is intended to be executed from cron at a regular interval (I use one hour). It will copy any system file which has been modified to a separate location and then (optionally) check the file into a source control system.

Most systems have either SCCS or RCS available, but not always both. In order to enable the same makefile to be used on every system in a heterogeneous environment, it has been written so the decision of what source control system to use is made at runtime. If directory directory named "RCS" exists, the RCS system will be used, and if the a directory named "SCCS" exists, the SCCS system will be used. If neither of

---

0. Network Information Service – formerly known as Yellow Pages

these directories exists, files will still be copied, but no version control system will be invoked.

This system will, of course, not tell who made any given changes, and neither will it have any information as to why a change has been done. It should therefore not necessarily be seen as a replacement for a procedure where the system administrators track such information, whether with a version control system, or by any otHER means. It does, however, provide an easy method for automatically tracking changes to important system files; it also provides a faster and more convenient way to restore old versions of a given file.

3. Assist in named configuration file maintenance

Another place where make(1) can participate in automating some of the work a system administrator has to do is in the maintenance of the configuration files for the domain name server (*named(8)*). Whenever a configuration file for the name server is modified, its sequence number must be incremented. If the person making the changes forgets to do so, then the change will not propagate properly to the other name servers.

The *makefile* in Appendix B is primarily used as a tool to handle the update of the serial numbers. However, it also has a number of other entries, which simplify other common tasks, mostly by reducing the amount of typing which is necessary. One example is whenever the name server is restarted with this make script, any messages it sends to syslogd will be shown on the screen. This minimizes the risk of an system administrator making a change and failing to notice any problems arising from the change.

With this makefile, the *named* data files, which are updated by the system administrator, are kept in */etc/named* and all begin with the prefix "db." (e.g. db.127.0.0). Whenever the system administrator checks the file out for update, RCS will automatically update the the version number. After the file has been updated, a call of make (without any argument) will check the file in under RCS (and the caller be prompted for comments on the change) and then copy the file to */etc/namedb/dbdir*, while removing the preceding "$Revision:" from the RCS revision number.

One word of caution: the domain name server has its own special interpretation of the dot in the serial number. When the name server loads the data files, it will substitute any dots in the sequence numbers with "000". Therefore "1.1" becomes "10001" and "1.11" becomes "100011". This is not a problem, as long as the major revision number is unchanged. However, if the current version number is 1.4593, and the major release number is bumped, so the new release number becomes 2.1, then, according to the name server, the old version number where "10004593", while the new number is "20001", and therefore considered very old and out of date. As an alternate method, the sequnce number can be based on the date. It requires a little more elaborate edition of the RCS provided date field, but is otherwise little different from the current startegy.

4. Maintain local and network wide files

There are a number of files on a UNIX system which simultaneously contain data specific to the local system and data of a more generic nature which applies to a large number of systems. Because of this split nature, it is not possible to maintain the file from a central location and distribute it as appropriate. The following examples show three different ways of solving this problem.

4.1 Maintain network wide content of /etc/motd

While the common usage of X on workstation and X terminals has caused most users to stay logged in for days at a time, it has greatly diminished the usage of putting system related information in the *message of the day* file (*/etc/motd*). However, it is still good practice to notify users of recent or upcoming changes to the system. In order to make this manageable at all, it will need to be possible to centralize the maintenance to a large degree.

The small makefile in Appendix C expects the information in the *motd* file to be kept in three files. The first, *motd.loc*, contains information which is only relevant to the local system. The *motd.net* contains information which is relevant to multiple systems, and therefore is maintained centrally and distributed in an adequate fashion, with a tool such as *rdist(1) fdist*, or *track*. Finally, information about the kernel (BSD style) is kept in the file *motd.ker*. If make is then called on an hourly basis, then it will rebuild the official

motd file whenever any of the three files mentioned above is modified.

4.2 Maintain network-wide content of /etc/group

The makefile discussed above is small and simple. However, if for some reason the concatenation fails, it is possible that the resulting file will be left empty. This may be acceptable for a file, such as */etc/motd*, whose purpose is purely of informative nature. However, if the file contains data which is used by the system in one form or another, an accidental truncation is not acceptable. The makefile in Appendix D rebuilds the */etc/group* file in a similar manner to the makefile for */etc/motd* in Appendix C, however, great care has been taken in this implementation, to ensure that if any step of the build process fails, then the original */etc/group* file is left in place without any changes.

This example's technique uses the the UNIX *ln(1)* command to implement a simple file locking. Traditionally, if a file is attempted to be linked to an already existing file, then the link program would fail and return an appropriate error code. However, in recent years, a number of UNIX system have been shipped where this command is severely broken and just overwrites the target file. The makefile in this example should not be installed on such systems, unless either the *ln(1)* command has been fixed or an alternative method of file locking implemented.

The file creation is performed in three steps:

1. First, a temp file is created and linked to the file */etc/group.new*. If this file has already been created by somebody else, then the link will fail, and make will return after removing the temporary file.
2. If the link completed successfully, we can remove the temporary file, as it is no longer needed. However, because of the manner this file was created in step 1, we are now sure to have a mutually exclusive access to that file.
3. The local- and network-wide content of the file is then copied into */etc/group.new*. This file now contains the new version of */etc/group*.
4. Finally, the old group file is moved out of the way, and the new one replaces it.

This method is as close to a fail safe implementation as is possible from the shell level.

4.3 Maintain network wide crontab files

Here's how to Maintain crontab for various system users (such as "root" "uucp", "usenet", "adm", etc). For each user, there will be a network-wide and an optional local component of the crontab file. This method is based on a System V style cron deamon, where each user has its own crontab file. There is no reason the principles discussed here could not be applied to a BSD style crontab file.

Acknowledgments

I thank Rob Kolstad for proofreading this paper.

Author Information

Bjorn Satdeva is the President of /sys/admin, inc., a consulting firm which specializes in Large Installation System Administration. Bjorn is also founder and former President of Bay-LISA, a San Francisco Bay Area user's group for system administrators of large sites, and Columnist for SysAdmin, a UNIX System Administration Magazine. Bjorn can be contacted by US Mail at /sys/admin, inc., 2787 Moorpark Avenue, San Jose, CA 95128, by e-mail at at bjorn@sysadmin.com, or by phone at (408) 241 3111.

References

1. Walter F. Tichy, RCS - A System for Version Control, Software--Practice & Experience 15, 7 (July 1985), pp 637-654.
2. Sccs, SunOS 4.x manpages.
3. Name Server Operations Guide for BIND

Appendix A:

Makefile to save copies of important system files. All such files are kept in the file List, with one file per line.

```
#
# Make copies of system configuration files.  All changes are placed
# under source control.
#

# Grrr ....
SHELL=      /bin/sh

.PRECIOUS:  ${SCCS_COPY} ${RCS_COPY} ${NONE_COPY}
.SILENT:

LIST= List

all:        ${LIST}
      -if [ -d RCS ] ; then TYPE=RCS ;
      elif [ -d SCCS ] ; then TYPE=SCCS;
      elif [ -d NONE ] ; then TYPE=NONE;
      else TYPE=NONE ; echo "Cannot determine sorce control mechanism" ;
      fi ;
      (for FILE in `cat ${LIST}` ; do
            COPY=`basename $${FILE}` ;
            if [ ! -f $${FILE} ] ; then continue ; fi ;
            make $${TYPE}  SRC=$${FILE} $${TYPE}_COPY=$${COPY} ;
      done ) | grep -v "is up to date" 2>&1 ; true


RCS:        ${RCS_COPY}
${RCS_COPY}:      ${SRC}
      touch ${RCS_COPY}
      if diff ${SRC} ${SCCS_COPY} > /dev/null 2>&1 ; then
            true ;
      else
            echo Saving ${SRC} on `hostname` ;
            if [ ! -f RCS/${RCS_COPY},v ] ; then
                  ci -q ${RCS_COPY} </dev/null ;
            else
                  co -f -l -q ${RCS_COPY} ;
                  cp ${SRC} ${RCS_COPY} ;
                  ci -l -q ${RCS_COPY} < /dev/null ;
            fi ;
      fi
```

```
SCCS:       ${SCCS_COPY}
${SCCS_COPY}:     ${SRC}
      touch ${SCCS_COPY}
      if diff ${SRC} ${SCCS_COPY} > /dev/null 2>&1 ; then
            true ;
      else
            echo Saving ${SRC} on `hostname` ;
            if [ ! -f SCCS/s.${SCCS_COPY} ] ; then
                  sccs create ${SCCS_COPY}  > /dev/null 2>&1 ;
            fi ;
            sccs get -e -s ${SCCS_COPY} ;
            cp ${SRC} ${SCCS_COPY} ;
            sccs delget -s ${SCCS_COPY} < /dev/null ;
            sccs get -s ${SCCS_COPY} ;
      fi

NONE:       ${NONE_COPY}
${NONE_COPY}:     ${SRC}
      touch ${NONE_COPY}
      if diff ${SRC} ${NONE_COPY} > /dev/null 2>&1 ; then
            true ;
      else
            echo Saving ${SRC} on `hostname` ;
            cp ${SRC} ${NONE_COPY} ;
      fi


${LIST}:
      @echo "Save file make on "`hostname`": ${LIST}: No such file"
      @false

FRC:
```

Appendix B:

```
#
# Makefile for maintaining the named database files (and specially to
# ensure the update of the files serial number
#

.SILENT:

# The location of the data files
DBDIR=      dbdir

# The name of active RCS file
RCS=  ${SRC:S;$;,v;g:S;^;RCS/;g}

# The name of the target data file
DB=  ${SRC:S;^;${DBDIR}/;g}

all:  update reload

# Reload the name server
reload::
      kill -HUP `cat /var/run/named.pid`
      sleep 1
      tail /var/log/messages | grep named

# Kill and restart the name server
restart::
      kill -9 `cat /var/run/named.pid`
      named
      sleep 1
      tail /var/log/messages | grep named

# Kill the name server
kill::
      kill -9 `cat /var/run/named.pid`

# Dump the internal databse from the name server
dump::
      kill -INT `cat /var/run/named.pid`
      sleep 1
      tail /var/tmp/named_dump* | grep named


# Dump the internal databse from the name server
update::
      (for FILE in db.* ; do make copy SRC=$${FILE} ; done )

copy::            ${DB}
${DB}::           ${RCS}
      cd ${DBDIR} ; co -p ../${RCS} |
            sed -e 's/\\$\$Revision: /    /' -e 's/\\$\$//' > ${SRC}

${RCS}::          ${SRC}
      echo RCS = ${RCS}
      ci -u ${SRC}
```

```
        touch ${RCS}
```

Appendix C:

```
.SILENT:

# Grrr .....
SHELL=              /bin/sh


all:        motd

#
# Update the Messages-Of-The-Day file
#
motd!       /etc/motd
/etc/motd:  /etc/motd.ker /etc/motd.net /etc/motd.loc
      cat /etc/motd.ker /etc/motd.net /etc/motd.loc > /etc/motd
```

Appendix D:

```
#
#
#

.SILENT:

# Grrr .....
SHELL=              /bin/sh

# Tmp file for group
TMP!=       echo "/etc/group.$$$$"

#
#

all:        group

#
# Update the group file
#
group!              /etc/group
/etc/group: /etc/group.loc /etc/group.net
            touch ${TMP}
            ln ${TMP} /etc/group.new || (rm ${TMP}; false)
            rm -f ${TMP}
            cat /etc/group.loc /etc/group.net |
                                sort -t: -n -k3,4 > /etc/group.new
            rm -f /etc/group.old
            mv /etc/group /etc/group.old; mv /etc/group.new /etc/group
```

Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection

Gene H. Kim*and Eugene H. Spafford
COAST *Laboratory*
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398

ABSTRACT

Tripwire is an integrity checking program written for the UNIX environment. It gives system administrators the ability to monitor file systems for added, deleted, and modified files. Intended to aid intrusion detection, Tripwire was officially released on November 2, 1992. It is being actively used at thousands of sites around the world. Published in volume 26 of `comp.sources.unix` on the USENET and archived at numerous FTP sites around the world, Tripwire is widely available and widely distributed. It is recommended by various computer security response teams, including the CERT and CIAC.

This paper begins by motivating the need for an integrity checker by presenting a hypothetical situation any system administrator could face. An overview of Tripwire is then described, emphasizing the salient aspects of Tripwire configuration that supports its use at sites employing modern variants of the UNIX operating system. Experiences with how Tripwire has been used in "in the field" are then presented, along with some conjectures on the prevalence and extent of system breakins. Novel uses of Tripwire and notable configurations of Tripwire are also presented.

1 Introduction

Tripwire is an integrity checking program written for the UNIX environment that gives system administrators the ability to monitor file systems for added, deleted, and modified files. Intended to aid intrusion detection, Tripwire was officially released on November 2, 1992,[1] and is being actively used at thousands of sites around the world. Published in volume 26 of `comp.sources.unix` and archived at numerous FTP sites around the world, Tripwire is widely available and widely distributed. It is now recommended by many computer security response teams, including the ARPA Computer Emergency Response Team (CERT).

Testing of Tripwire started in September 1992. Since then, its design and code have been available for scrutiny by the public at large. The design and implementation are described in detail in [6].

*Gene Kim is currently at the University of Arizona.

[1] That release date was chosen for its historical significance as well as being convenient to our development schedule.

An intensive beta test period resulted in Tripwire being ported to over two dozen variants of UNIX, including several versions neither author had ever encountered. Currently entering its seventh (and possibly last) revision, Tripwire has met our design goals of being sufficiently portable, scalable, configurable, flexible, extensible, secure, manageable, and malleable to enjoy widespread use.

This paper documents some of our experiences and discoveries based on our development and use of Tripwire. It begins by motivating the use of an integrity checking tool (such as Tripwire) through the presentation of a hypothetical scenerio that a UNIX system administrator could face. Next, we present an overview of Tripwire's design, emphasizing the salient configuration aspects that allow its use in modern UNIX variants. We then discuss experiences gathered from Tripwire users since its September 1992 release. These stories seem to confirm the practicality of this integrity checking scheme. There are at least seven documented cases of Tripwire notifying system administrators of intruders' system tampering. We present our conjectures on the prevalence and extent of system breakins based on our data. We also describe novel uses of Tripwire and surprising configurations that have been reported to us. Feedback that has shaped the direction of Tripwire development is also presented.

Tripwire stands as an example how a simple idea can be developed into a general and effective tool to enhance UNIX security while also posing almost no threat to the systems under guard. Unlike programs like password crackers or flaw probes, Tripwire cannot be turned against a system to identify or exploit weaknesses or flaws. It is also an example of how a program may have uses unanticipated by its authors.

2 Motivation

2.1 A cautionary tale[2]

Ellen runs a network of 50 networked UNIX computers representing nearly a dozen vendors — from PCs running Xenix to a Cray running Unicos. This morning, when she logged in to her workstation, Ellen was a bit surprised when the `lastlog` message indicated that root had logged into the system at 3 AM. Ellen thought she was the only one with the root password. Needless to say, this was not something Ellen was happy to see.

A bit more investigation revealed that someone — certainly not Ellen — had logged on as root, not only on her machine but also on several other machines in her company. Unfortunately, the intruder deleted all the accounting and audit files just before logging out of each machine. Ellen suspects that the intruder (or intruders) ran the compiler and editor on several of the machines. Being concerned about security, Ellen is worried that the intruder may have thus changed one or more system files, thus enabling future unauthorized access as well as compromising sensitive information. How can she tell which files have been altered without restoring each system from backups?

Poor Ellen is faced with one of the most tedious and frustrating jobs a system administrator can have — determining which, if any, files and programs have been altered without authorization. File modifications may occur in a number of ways: an intruder, an authorized user violating local policy or controls, or even the rare piece of malicious code altering system executables as others are run. It might even be the case that some system hardware or software is silently corrupting vital system data.

In each of these situations, the problem is not so much knowing that things might have been changed; rather, the problem is verifying exactly which files — out of tens of thousands of files in dozens of gigabytes of disk on dozens of different architectures — might have been changed. Not only is it necessary to examine every one of these files, but it is also necessary to examine directory information as well. Ellen will need to check for deleted or added files, too. With so many different systems and files, how is Ellen going to manage the situation?

Resolving such a situation would prove tedious and labor-intensive for even the most well-

[2] This is taken from [7].

prepared system administrator. Consider the problems facing system administrators who use simple checklisting schemes:

2.2 The resulting challenges

Established techniques for monitoring file systems for potentially dangerous changes include maintaining checklists, comparison copies, checksum records, or a long history of backup tapes for this kind of contingency [4, 2]. However, these methods are costly to maintain, prone to error, and susceptible to easy spoofing by a malicious intruder.

For instance, the UNIX utility `find(1)` is often used to generate a checklist of system files, perhaps in conjunction with `ls(1)`. This list is then saved and compared using `diff(1)` to determine which files have been added or deleted, and to find which files have conflicting modification times, ownership, or sizes. An added level of security could be added by augmenting these lists with information from `sum(8)` or `cksum(8)`, as is done by the *crc_check* program included with COPS [3].

However, numerous shortcomings in these simple checklisting schemes prevent them from being completely trustworthy and useful. First, the list of files and associated checksums may be tedious to maintain because of its size. Second, using timestamps, checksums, and file sizes does not necessarily ensure the integrity of each file (e.g., once intruders gain root privileges, they may alter timestamps and even the checklists at will). Furthermore, changes to a file may be made without changing its length or checksum generated by the `sum(8)` program. And this entire approach presumes that `ls(1)`, `sum(8)`, and the other programs have not been compromised! In the case of a serious attack, a conscientious administrator needs stronger proof that important files have remained unchanged. But what proof can be offered that is sufficient for this situation?

2.3 The resulting wishlist

A successful integrity checking scheme requires a high level of automation — both in generating the output list and in generating the input list of files. If the scheme is difficult to use, it may not be used often enough — or worse, used improperly. The automation should include a simple way to describe portions of the filesystem to be traversed. Additionally, in cases where files are likely to be added, changed, or deleted, it must be easy to update the checklist database. For instance, files such as `/etc/motd` may change daily or weekly. It should not be necessary to regenerate the entire database every time this single file changes to maintain database accuracy.

Ideally, our integrity checking program could be run regularly from `cron(8)` to enable detection of file changes in a timely manner. It should also be possible to run the program manually to check a smaller set of files for changes. As the administrator is likely to compare the differences between the "base" checklist and the current file list frequently, it is important that the program be easy to invoke and use.

A useful integrity checker must generate output that is easy to scan. A checker generating three hundred lines of output from each machine for the system administrator to analyze daily would be self-defeating — this is far too much to ask of even the most amazingly dedicated system administrator! Thus, the program must allow the specification of filesystem "exceptions" that can change without being reported, and hence reduce "noise." For example, changes in system log file sizes are expected, but a change in inode number, ownership, or file modes is cause for alarm. However, a change in any value stored in the inodes (except for the access timestamp) for system binaries in `/bin` should be reported. Properly specified, the integrity checker should operate unobtrusively, notifying Ellen when a file changes outside the specified bounds, and otherwise running quietly.

Finally, assuming that Ellen wants to run the integrity checker on every machine in her network, the integrity checker should allow the reuse and sharing of configuration files wherever possible. For example, if Ellen has twenty identical workstations, they should be able to share a common configuration file, but allowing machine-specific oddities (i.e., some software package installed on

only one machine). The configuration should thus support selective reuse to reduces the opportunity for operator error.

3 Tripwire design

The criteria we describe above represent the motivation for some of the key design issues behind Tripwire. Ultimately, the goal of Tripwire is to detect and notify system administrators of changed, added, and deleted files in some meaningful and useful manner. However, the success of such a tool depends on how well it works within the realities of the administration environment. This includes appropriate flexibility to fit a range of security policies, portability to different platforms in the same administrative realm, and ease of use.

3.1 Component overview

A high level model of Tripwire operation is shown in Figure 1. This shows how the Tripwire program uses two inputs: a *configuration* describing the file system objects to monitor, and a *database* of previously generated signatures putatively matching the configuration.

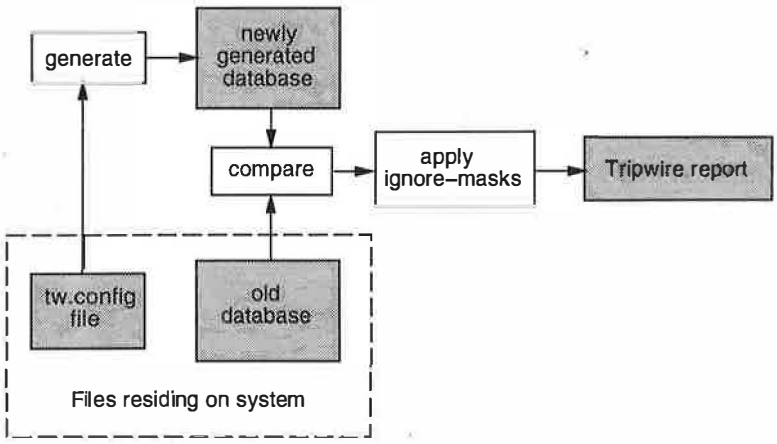


Figure 1: Diagram of high level operation model of Tripwire

In its simplest form, the configuration file contains a list of files and directories to be monitored, along with their associated selection mask (i.e., the list of attributes that can be safely ignored if changed). The database file is generated by Tripwire, containing a list of entries with filenames, inode attribute values, signature information, selection masks, and the configuration file entry that generated it.

3.2 Modes of Operation

In the four (mutually exclusive) modes of Tripwire operation, program operation is driven by the contents of the configuration file, `tw.config`. Each mode is described in turn below.

In *database initialization* mode, Tripwire generates a baseline database containing entries for every file specified in the configuration file, `tw.config`. Each database entry contains the filename, inode attributes, signature information, its selection mask, and the configuration entry that generated it. As entries in `tw.config` can be directories, each entry can map to many entries in the database.

In *integrity checking* mode, Tripwire reads the `tw.config` file and generates a new database. Tripwire then compares this database with the baseline, producing a list of added and deleted files. For those files that have changed, the selection mask is applied to determine whether a report should

be generated. Note that the selection mask stored in the baseline database is used, not the one in `tw.config`, based on the premise that the base database has been stored on some secure media (e.g., read-only floppy).

When files change for legitimate reasons and no longer match the baseline database description, updating the baseline database becomes necessary. Tripwire offers two modes to ensure database consistency. In *database update* mode, Tripwire is given a list of files or configuration entries on the command line. The database entries for these files are regenerated, and a new database written out. Tripwire then instructs the system administrator to move this database to secure media. In *interactive database update* mode, Tripwire first generates a list of all changes (ala integrity checking mode). For each of these changes, Tripwire then asks the system administrator whether the specified file or entry should be updated.

3.3 Scalability aids

Tripwire includes an M4-like preprocessing language [5] to help system administrators maximize reuse of configuration files. By including directives such as "`@@include`", "`@@ifdef`", "`@@ifhost`", and "`@@define`", system administrators can write a core configuration file describing portions of the file system shared by many machines. These core files can then be conditionally included in the configuration file for each machine.

To allow the possible use of Tripwire at sites consisting of thousands of machines, configuration and database files do not need to reside on the actual machine. Input can be read from file descriptors, open at the time of Tripwire invocation. These file descriptors can be connected to UNIX pipes or network connections. Thus, a remote server or a local program can supply the necessary file contents. Supporting UNIX style pipes also allows for outside programs to supply encryption and compression services — services that we do not anticipate including as a standard part of the core Tripwire package.

Tripwire does not encrypt the database file so as to ensure that runs can be completely automated (i.e., no one has to type in the encryption key every night at 3 AM). Because the database contains nothing that would aid an intruder in subverting Tripwire, this does not undermine the security of the system. However, if Tripwire is used in an environment where the database is encrypted as a matter of policy, the interface supports this, as described above.

3.4 Configurability aids

Tripwire makes a distinction between the configuration file and the database file. Each machine may share a configuration file, but each generates its own database file. Thus, identically configured machines can share their configuration database, but each has its integrity checked against a per-machine database.

Because of the preprocessor support, system administrators can write Tripwire configuration files that support numerous configurations of machines. Uniform and unique machines are similarly handled. This helps support reuse and minimize user overhead in installation.

The configuration file for Tripwire, `tw.config`, contains a list of entries, enumerating the set of directory (or files) to be monitored for changes, additions, or deletions. Associated with each entry is a selection-mask (described in the next section) that describes which file (inode) attributes can change without being reported as an exception. An excerpt from a set of `tw.config` entries is shown in Figure 2.

Prefixes to the `tw.config` entries allow for pruning (i.e., preventing Tripwire from recursing into the specified directory or recording a database entry for a file). Both inclusive and non-inclusive pruning are supported; that is, a directory's contents only may be excluded from monitoring, or the directory and its contents may both be excluded.

By default, all entries within a named directory are included when the database is generated.

```
# file/dir      selection-mask
/etc            R       # all files under /etc
@@ifhost solaria.cs.purdue.edu
 !/etc/lp               #  except for SVR4 printer logs
@@endif
/etc/passwd     R+12    # you can't be too careful
/etc/mtab       L       # dynamic files
/etc/motd       L
/etc/utmp       L
=/var/tmp       R       # only the directory, not its contents
```

Figure 2: An excerpt from a `tw.config` file

Each entry is recorded in the database with the same flags and signatures as the enclosing, specified directory. This allows the user to write more compact and inclusive configuration files. Some users have reported using configuration files of a simple /, naming all entries in the file system!

The `tw.config` file also contains the names of files and directories with their associated selection-mask. A selection-mask may look like: `+pinugsm12-a`. Flags are added ("+") or deleted ("-") from the set of items to be examined.

Tripwire reads this as, "Report changes in permission and modes, inode number, number of links, user id, group id, size of the file, modification timestamp, and signatures 1 and 2. Disregard changes to access timestamp."

A flag exists for every distinct field stored in an inode. Provided is a set of templates to allow system administrators to quickly classify files into categories that use common sets of flags:

- **read-only files** Only the access timestamp is ignored.
- **log files** Changes to the file size, access and modification timestamp, and signatures are ignored.
- **growing log files** Changes to the access and modification timestamp, and signatures are ignored. Increasing file sizes are ignored.
- **ignore nothing** self-explanatory
- **ignore everything** self-explanatory

Any files differing from their database entries are then interpreted according to their selection-masks. If any attributes are to be monitored, the filename is printed, as are the expected and actual values of the inode attributes. An example of Tripwire output for changed files is shown in Figure 3.

A "quiet option" is also available through a command-line option to force Tripwire to give terse output. The output when running in this mode is suitable for use by filter programs. This allows for an external script to execute automated actions if desired. One example would be to use the terse output of Tripwire after a breakin to quickly make a backup tape of only changed files, to be examined later.

By allowing reporting to be dictated by local policy, Tripwire can be used at sites with a very broad range of security policies.

```
changed: -rw-r--r-- root           20 Sep 17 13:46:43 1993 /.rhosts
### Attr        Observed (what it is)        Expected (what it should be)
### ============ ============================== ============================
/.rhosts
st_mtime:       Fri Sep 17 13:46:43 1993      Tue Sep 14 20:05:10 1993
st_ctime:       Fri Sep 17 13:46:43 1993      Tue Sep 14 20:05:10 1993
```

Figure 3: Sample Tripwire output for a changed file

3.5 Signature support

Tripwire has a generic interface to "signature[3]" routines and supports up to ten signatures to be used for each file. The following default methods are included in the latest Tripwire distribution: MD5[10] (the RSA Data Security, Inc. MD5 Message-Digest Algorithm), MD4[9] (the RSA Data Security, Inc. MD4 Message-Digest Algorithm), MD2 (the RSA Data Security, Inc. MD2 Message-Digest Algorithm),[4] Snefru[8] (the Xerox Secure Hash Function), and SHA (the NIST proposed Secure Hash Algorithm). Tripwire also includes POSIX 1003.2 compliant CRC-32 and CCITT compliant CRC-16 signatures.

Each signature may be included in the selection-mask by including its index. Because each signature routine presents a different balance in the equation between performance and security, the system administrator can tailor the use of signatures according to local policy. By default, MD5 and Snefru signatures are recorded and checked for each file. However, different signatures can be specified for each and every file. This allows the system administrator great flexibility in what to scan, and when.

Also included in the Tripwire distribution is **siggen**, a program that generates signatures for the files specified on the command line. This tool provides a convenient means of generating any of the included signatures for any file.

The code for the signature generation functions is written with a very simple interface. Thus, Tripwire can be customized to use additional signature routines, including cryptographic checksum methods and per-site hash-code methods. Tripwire has room for 10 functions, and only seven are preassigned, as above.

4 Experiences

Since the initial Tripwire release in November 1992, seven subsequent versions have been released to incorporate bug fixes, support additional platforms, and add new features. The authors estimate Tripwire is being actively used at several thousand sites around the world. Retrievals of the Tripwire distribution from our FTP server initially exceeded 300 per week. Currently, seven months after the last official patch release, we see an average of 25 fetches per week. This does not include the copies being obtained from the many FTP mirror sites around the net.

We have received considerable feedback on Tripwire design, implementation, and use. We believe that version 1.1 of Tripwire has succeeded in meeting most of the goals of system administrators needing an integrity checking tool.

In this section, we present feedback from system administrators that seem to validate the workability of integrity checkers and present conjectures on the prevalence and extent of system breakins. We also prevent novel uses of Tripwire and surprising configurations that have been

[3] We use the term *signature* to include checksums, message digests, secure hash functions, and/or cryptographic signatures.

[4] The copyright on the available code for MD-2 strictly limits its use to privacy-enhanced mail functions. RSA Data Security, Inc. has kindly given us permission to include MD-2 in the Tripwire package without further restriction or royalty.

reported to us. Feedback that has shaped the direction of Tripwire development is also presented.

4.1 First, the good news...

We have gathered reports of at least seven cases where Tripwire has alerted system administrators to intruders tampering with their systems. In at least two of these cases, the penetration was widespread, with numerous system programs and libraries replaced with trojan horses.

Potentially less exciting than these stories, but equally inspiring, are the dozens of stories we have received of sites using Tripwire as a system administration enforcement tool. System administrators report having found hundreds of program binaries changed, only to find that another person with system-level access had made the changes without following local notification policy.

There has also been one reported case of a system administrator detecting a failing disk with Tripwire. The normal system logging reporting the failure was not read very often by the system administrator, but the Tripwire output was surveyed daily.

All three classes of stories validate the theory behind integrity checking programs. Although the foundations of integrity checkers in UNIX security have been discussed in [1, 2, 4], when Tripwire design was started in May 1992, no usable, publically available integrity tools existed — providing one of the primary motivations for writing Tripwire.

4.1.1 Where are all the bad guys?

The dramatically increased number of network breakins throughout the Internet in early 1994 presented an opportunity to compare the prevalence of system breakins at sites running Tripwire with those sites that did not.[5]

One of us (Spafford) posted a query on USENET asking whether any sites running Tripwire were successfully subverted as described in the CERT advisory. Surprisingly, no system administrator at any site reported such a breakin. Why? Furthermore, out of the thousands of machines running Tripwire, why have we heard of only seven Tripwire-discovered breakins since 1992? We offer some possibilities:

- The intruders have given up: if the competence and ambition of intruders have dropped since 1992, the small number of reported incidents could be explained away. However, this is not consistent with the reports from response teams and the frequent advisories reporting newly discovered system vulnerabilites being exploited by intruders.
- The sites running Tripwire are not interesting: if sites running Tripwire offer nothing of interest to an intruder, then one would expect few breakin attempts. However, given that some of the highest-profile UNIX sites in the nation (e.g., public access UNIX sites, government information servers, military sites) are running Tripwire, this seems implausible.
- The site admins are not telling: it may be the case that system administrators at sites where breakins have occurred are not willing to tell us that they have been sucessfully attacked. This is possible considering the nature of many of the sites running Tripwire, but we would expect at least a few reports to be made in confidence.
- The sites are more security-conscious: if system administrators running Tripwire are also considerably more successful in securing their systems than other UNIX sites, intruders would find known vulnerabilities corrected, greater than usual protection measures employed, and more vigilance from system administrators. This would explain the low number of reported intrusion incidents from a sample made up exclusively of sites running Tripwire.
- The sites have already been attacked: the Tripwire baseline database should be generated from a clean distribution; one that is assured to be free of trojan horses, logic bombs, etc.

[5] See, in particular, CERT advisory CA-94:01 dated February 3, 1994.

This usually means reinstalling the operating system from vendor-supplied media. However, this is often prohibitively inconvenient. If databases are being generated on machines already compromised, then Tripwire will have been installed too late to have reported those critical file tamperings. If this is the case, then many sites have not reported breakins because they continue to be unaware of them.

- The intruders have completely subverted integrity checking schemes: changed files are usually detected through the use of file signatures. An intruder could be modifying files in such a way that the resulting files preserve their original signatures. However, Tripwire includes seven signature routines, and the choice of which signatures are used for any file is not fixed. That an intruder could be using such a technique is possible, but the possibility is so small as to be almost nonexistant.

Of the conjectures offered, the supposition most credible is that system administrators who run Tripwire represent a poor sample for determining system breakins. A substantial portion of the thousands of Tripwire e-mail messages we have received underscore the competence and paranoia of these system administrators.

However, despite these system administrators' best intentions, their lack of available, trusted signatures for critical operating system files is especially noteworthy. Instead of installing a system from distribution media, they typically choose a machine that they believe is "clean," using it to generate the baseline database. This assumption, however, may be completely ungrounded. We know of at least one case of a reported breakin at a site where system administrators discovered that their "safe baseline" was actually the first to be attacked.

We believe that operating system distributions, and perhaps other software (e.g., compilers, system administration tools), should be shipped with a complete signature database. This information could then be stored locally on some secure media or offline, and then used in the event of a suspected breakin.

Not coincidentally, Tripwire could be used effectively in such a role if software vendors supplied a Tripwire-conformant database with their distributions. The Tripwire database is suited for such a purpose: it is ASCII, mostly human readable, simple to parse and construct (21 fixed fields), compact (signatures are stored in base 64), self-contained (all the database information is encapsulated in a single file), and individual entries can be checked using the `siggen` program.

4.1.2 How about the good guys?

"The mark of a good tool is that it is used in ways that its author never thought of." [6]

As noted previously, many system administrators are using Tripwire primarily as a tool to enforce local policy. When system administration duties are delegated among numerous people, changes made by one person often go unnoticed and unexplained to others. Running Tripwire allows these changes to be noticed in a timely manner — a goal very similar to intrusion detection.

Another application we note uses Tripwire to help salvage file systems not completely repaired by `fsck`, the program run at system boot that ensures consistency between file data and their inodes. In cases when file blocks cannot be bound to its file name, they are placed in the `lost+found` directory and renamed to some (less than useful) number. If a database of file signatures is available, this file could be rebound to its original name by searching the database for a matching signature.

Because providing a useful tool to system administrators was one of the goals of writing Tripwire, the variety of applications of Tripwire outside the domain of intrusion detection has been especially surprising and satisfying for us. We are still collecting other stories of novel use of the Tripwire package.

[6] Brian Kernighan has said this, in one form or another, in several of his presentations and written works. This particular version was in private e-mail to one of us in response to a citation request.

4.2 Stealth-Tripwire

Several site administrators have reported going to considerable lengths to conceal the operation of Tripwire. These system administrators feel strongly that they should not advertise their security measures or policies.

As a result, Tripwire is not being run through programs like `cron(8)`, the conventional means of executing programs on a regular schedule. Instead, a wide variety of local tools are used. For example, a special daemon might be loaded at system startup, waking only to run Tripwire at a scheduled time.

Where `cron` is used, deception through indirection is sometimes used to prevent an intruder from immediately detecting evidence of Tripwire operation. In one case, a system administrator uses three levels of indirection before finally executing Tripwire (e.g., `cron` runs a script that runs a script than runs a script that runs Tripwire).

We wonder whether these measures to conceal Tripwire are necessary, or even desirable. One of us (Spafford) has heard of an "underground" publication warning of the need for special vigilance when attempting to crack systems running Tripwire. If this warning is heeded, then the presence of Tripwire may have the ability to deter crackers. Advertising the use of Tripwire (even if not true) could thus help avert attacks.

4.3 Security is nice, if it's not too difficult

Because Tripwire reports are only as reliable as its inputs, the design document stresses the need to ensure the integrity of the baseline database. Thus, we suggest that the baseline database be moved to some secure read-only media immediately after it is is generated.

The most common Tripwire configuration reported to us to facilitate this is the use of a "secure server," a specialized server receiving extra scrutiny from administrators. A remote file system or server process is then used to export the baseline database to clients.

However, several sites have gone to much further lengths to maintain the integrity of Tripwire databases. At least two sites have considerably modified Tripwire to support alternate channels for receiving the database and transmitting the report, adding layers for networking support, encryption, and host authentication.

Since its original release, we have added full support for using open UNIX file descriptors to read the Tripwire configuration and database files. This allows system administrators to easily add support for encryption and compression without having to modify the Tripwire package so drastically. Instead, a wrapper program (even a shell script) can be used to supply these facilities. Used with named pipes, wrapper scripts in Perl or Tcl, or simple network clients this also allows centralized administration of Tripwire checks in large installations.

It is interesting to note that mistrust of networked file systems has motivated many of the enduser-modifications to Tripwire. However, some of the replacements we have have been told about sound as if they include other weaknesses. A sound, portable solution to the problem has yet to appear.

4.4 Paranoia is unbounded

The Tripwire design document recommends running Tripwire in integrity checking mode on a regular basis (e.g., daily) to ensure that file system tampering can be detected in a timely manner.

However, there have been two reported cases of large sites running Tripwire far more frequently. In fact, these experiences motivated the option of including a signature selection feature to allow skipping certain signatures by specifying choices on the command line. Because these site admins were running Tripwire on their machines *hourly* with all signature checking enabled, the Tripwire runs were not completed by the time the next Tripwire run started!

We were left wondering what these machines did besides spending all the CPU cycles computing file signatures. We also wonder why they placed so little faith in their other security measures, and what level of threat they were actually fearing.

In contrast is the lack of use of an ideal Tripwire-aided bit of paranoia. One of the ideas behind Tripwire's design (and the name itself) was for system managers to scatter "plant" files on their system, similar to what was done by Cliff Stoll[11]. These files would have interesting names (e.g., `master-passwords`), but useless contents. These files would not normally be accessed by users, but might be prime targets for intruders. By monitoring these files as "mini-tripwires," it would be possible to detect an otherwise stealthy intrusion. We have yet to hear of anyone using this scheme to good effect.

4.5 Scalability includes sites large and small

When designing Tripwire, we were more concerned about the problems facing system administrators at large sites. Although design considerations were made for these configurations, how Tripwire was used at small sites was more surprising.

4.5.1 Mega-Tripwire

Tripwire provides a configuration language intended to aid system administrators in managing larger sites. We were especially interested in how these tools would be used by system administrators – the Tripwire design document suggests that a core configuration file could be shared by numerous hosts by using the `@@include` directive.

From reports we have gathered, this appears to be a less than popular method. Instead, system administrators create one configuration file to be shared by all machines, using the `@@ifhost` directive to segregate non-common file groups.

We suspect that the overhead of tracking multiple configuration files outweighs the inconvenience caused by files obfuscated by many "`@@ifdef`" statements. These shared configuration files are apparently still manageable, as the number of entries in the file is usually not large. (We suspect that if files had to be individually enumerated, these configuration files would be far larger, and therefore unmanageable.)

Tripwire has proven scalable, with documented cases of sites of almost one thousand machines running Tripwire, as well as sites of only one machine. That system administrators have done so using a different mechanism than suggested in the design document is especially interesting. That system administrators are not slavishly following our design document is reassuring.

4.5.2 Micro-Tripwire

How Tripwire is used on workstations with minimal disk resources proved surprisingly elegant. Although the Tripwire configuration file allows considerable flexibility in specifying files and directories to monitor, configuration files concocted by system administrators for these workstations consist of only one character: "/"

Thusly, Tripwire scans all the local disk partitions under the root directory, collecting the default MD5 and Snefru signatures. For some sites, this has proved adequate for all their machines!

4.6 Running Tripwire on the Sasquatch Kumquat Mark VIIa/MP

Tripwire has proven to be highly portable, successfully running on over 28 UNIX platforms. Among them are Sun, SGI, HP, Sequents, Pyramids, Crays, Apollos, NeXTs, BSDI, Lynix, Apple Macintosh, and even Xenix. Configurations for new operating systems has proven to be sufficiently general to necessitate the inclusion of only eight example `tw.config` files.

However, potentially challenging situations result when we receive requests from system administrators asking for help compiling Tripwire on machines that neither of us have ever heard of.

In one case, this was a machine only sold in Australia and shipped with incorrect system libraries. Other instances included an especially ignoble machine that has not been sold since 1986 (predating college for of us), and numerous machines with non-standard compilers, libraries, system calls, and shells.

In all but two cases (of the last variety), we have incorporated changes in Tripwire sources to accomodate these machines. In most cases, there has been a sufficiently large group of system administrators with similarly orphaned machines who put together a suitable patch to allow correct Tripwire compilation and operation.

It is interesting to analyze the time needed to fully support a configuration. Full support for Sun's new Solaris operating system was added two months after the initial Tripwire release. A workaround for the two afforementioned Australian machines was released six months after the problems were first reported. However, some Tripwire users running machines from a large workstation vendor continue to be unable to find a compiler that correctly generates a Tripwire that passes the entire test suite; investigation has determined that this is because of non-standard and broken compilers and libraries on those platforms.

4.7 You added WHAT to Tripwire?

We recently received a report from a user who is adding support for Intel machines running UNIX to allow Tripwire to check mounted MSDOS file systems. In such a manner, they are using Tripwire to check not only UNIX file systems, but also their DOS files (for viruses, etc.).

We also received, and are incorporating into a future Tripwire patch release, a set of changes to allow Tripwire to check the integrity of symbolic links — a weakness noted in [12]. One novel and elegant solution was implemented by storing the contents of the symbolic link as a signature.[7] Our actual solution will involve taking the signatures of the link field contents.

We are especially pleased that system administrators can so easily make feature additions that they perceive as necessary. We believe this reflects well on the design and coding of the Tripwire release, although we realize that the code is rather opaque in many spots.

4.8 Static file systems aren't

According to system administrators, the ability to update Tripwire databases is among its most important features. Files seem to change for many unforeseen reasons. Consequently, the database is updated regularly. The addition of the interactive update facility in Tripwire was among the most enthusiastically received features.

Allowing database updates was a feature that we resisted for several months during the beta test period in 1992. We believed (and still do, in part) that ease of update may lead some administrators to be careless in their storage of the database, thus weakening the assurance Tripwire is capable of providing. That users acquiesced and still used Tripwire despite its lack of ability to update the baseline database without regenerating the entire database astounds the authors — in hindsight, at least.

5 Conclusions

Tripwire has proven to be a highly portable tool that system administrators can build using commonly available tools. It is completely self-contained, and once built, requires no other tools for execution. Tripwire is publically available, is widely distributed, and widely used.

Tripwire has been used by system administrators in large and small sites: we have documented Tripwire's active use at single machine sites, as well as sites having several hundreds of machines. We have yet to hear a report of a site where Tripwire was installed and then removed because it did not function according to expectation, or because it was too difficult to build or maintain. Coupled

[7] This solution was proposed and implemented by Paul Szabo of the University of Sydney.

with the many positive comments we have received, and the fact that Tripwire has already caught several intruders, leads us to conclude that our analysis and design are successful. We hope this effort serves as a model for others who consider building security tools with similar goals.

6 Availability

The beta version of Tripwire was made publically available and posted to `comp.sources.unix` on November 2, 1992 after three months of extensive testing. Over three hundred users around the world critiqued the four preliminary releases during Summer 1992, guiding the development towards a shippable, publically available tool. The formal release of Tripwire occurred in December of 1993.

Tripwire source is available at no cost.[8] It has appeared in `comp.sources.unix` (volume 26) on Usenet, and is available via anonymous FTP from many sites, including `ftp.cs.purdue.edu` in `pub/spaf/COAST/Tripwire`. Those without Internet access can obtain information on obtaining sources and patches via e-mail by mailing to `tripwire-request@cs.purdue.edu` with the single word "help" in the message body.

We regret that we do not have the resources available to make tapes or diskette versions of Tripwire for anyone other than COAST Project sponsors. Therefore, we ask that you not send us media for copies – it will not be returned.

References

[1] Vesselin Bontchev. Possible virus attacks against integrity programs and how to prevent them. Technical report, Virus Test Center, University of Hamburg, 1993.

[2] David A. Curry. *UNIX System Security: A Guide for Users and System Administrators.* Addison-Wesley, Reading, MA, 1992.

[3] Daniel Farmer and Eugene H. Spafford. The COPS security checker system. In *Proceedings of the Summer Conference*, pages 165–190, Berkely, CA, 1990. Usenix Association.

[4] Simson Garfinkel and Gene Spafford. *Practical Unix Security.* O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[5] Brian W. Kernighan and Dennis M. Ritchie. *The M4 Macro Processor.* AT&T Bell Laboratories, 1977.

[6] Gene H. Kim and Eugene H. Spafford. The design and implementation of tripwire: A file system integrity checker. Technical Report CSD–TR–93–071, Purdue University, nov 1993.

[7] Gene H. Kim and Eugene H. Spafford. Monitoring file system integrity on unix platforms. *InfoSecurity News*, 4(4):21–22, July 1993.

[8] Ralph C. Merkle. A fast software one-way hash function. *Journal of Cryptology*, 3(1):43–58, 1990.

[9] R. L. Rivest. The md4 message digest algorithm. *Advances in Cryptology — Crypto '90*, pages 303–311, 1991.

[10] R. L. Rivest. RFC 1321: The md5 message-digest algorithm. Technical report, Internet Activities Board, April 1992.

[11] Cliff Stoll. *The Cuckoo's Egg.* Doubleday, NY, NY, October 1989.

[12] David Vincenzetti and Massimo Cotrozzi. ATP anti tampering program. In Edward DeHart, editor, *Proceedings of the Security IV Conference*, pages 79–90, Berkeley, CA, 1993. USENIX Association.

[8] It is not "free" software, however. Tripwire and some of the signature routines bear copyright notices allowing free use for non-commercial purposes.

Installing and Managing Remote Sites

Scott Cohan <scohan@esm.com>
Director, Enterprise Systems Management Corporation

Steve Miano <stevem@esm.com>
Director, Enterprise Systems Management Corporation

Abstract

Any organization that uses computers must deal with (or struggle with) the basic tasks needed for system installation and management. On-site support personnel spend their time completing administrative tasks and implementing mechanisms to automate functions whenever possible. The mechanisms invented to automate common tasks are often creative and robust.

For organizations that bear responsibility for remote computer systems, new and innovative solutions are required. These remote sites are often numerous and smaller in size than their "home office" counterparts, yet the computer support team is expected to provide the same level of service. Unfortunately, most organizations attempt to apply the solutions that work locally to their remote sites, resulting in extended manpower ratios, or worse, a reduced level of support.

Problem Identification

In order to develop creative solutions for the unique problems of remote sites, one must first identify the issues that differentiate them from local sites. Unique characteristics of remote sites typically include:

- Significant distance from the main support office
- Smaller number of systems
- Lack of replacement parts
- Lack of manpower/expertise
- Differing time zones
- Less network bandwidth (wide-area vs. local-area)

Each of these characteristics presents new challenges:

- Long distances between home and remote sites typically means that a support person can't expect to get "hands-on" without some extended travel.
- The reduced number of systems creates a reluctance to invest significant money for support functions. Nonetheless, the ability to install and manage a small site may be lucrative for the company's business, and often leads to an expansion in the number of smaller locations.
- Providing high-availability of replacement hardware is a costly proposition that many companies opt to do without, at the risk of reduced service levels.
- On-site support personnel may not be available at small sites, with the end-user or customer often involved in the resolution of problems. At such sites additional expertise is required, and difficult to arrange.
- Differing time zones create logistical problems for system maintenance and scheduling of routine tasks. For example, if users in New York, London and Tokyo share a common database, finding a convenient time to reboot or backup that machine can be difficult.
- Lower network bandwidth requires increased attention to how systems communicate with each other, utilize the network, and provide redundancy.

Recommendations

Plan Ahead

When a remote system fails, fast and effective diagnosis will depend less on the measures taken ***after*** the failure, and more on measures taken during the design and implementation phases. Designing with remote management in mind is unquestionably the most important component of a successful remote infrastructure.

A popular advertisement uses the slogan "you can pay me now, or pay me later". This is especially true when dealing with remote sites. In the short term it may cost more to include redundant systems and networking equipment as part of the remote design, but the cost will be recovered quickly during the support phase of the system lifecycle.

Remote Access

The solution to many remote problems begins with access. If a support person can obtain full access to the remote equipment, she can diagnose and repair problems as if she were actually at the scene. A good design will guarantee diagnostic capability regardless of which system component fails.

Another advantage of remote access is that it allows your home-office experts to join in problem resolutions as if they were present at the remote site. In a single day you can provide a level of service which makes it look like your "superstars" were everywhere at once - even when the problem locations are thousands of miles apart.

Terminal Servers

For workstations and network equipment, diagnosis means finding a way to provide console access when a piece of hardware is unreachable via the normal network. Terminal servers such as the Cisco CS-500 can help provide this capability by allowing multiple serial connections. To guarantee their availability, the design should place terminal servers on separate redundant networks, with serial connectivity to the opposing networks.

In cases where a workstation or networking component fails, terminal servers allow the problem to be diagnosed and the repair coordinated remotely without unnecessary involvement by the end-user or customer. The result is faster, more economical remote administration.

Design Strategy

Proper design schemes for medium-sized sites can help to provide redundancy. The network and systems infrastructure should insure that the failure of any single component does not produce a condition that renders the entire remote site inaccessible.

In sites which contain multiple networks, terminal servers can be attached to one subnet while providing serial access to equipment on another. The consoles of routers, workstations and even the terminal servers themselves may be cross-connected onto alternate networks allowing repair during failures. Such designs use terminal servers and redundant network equipment to provide high-availability and remote diagnostic capability.

Finally, modems can provide access to remote sites during WAN outages. This is often a more economical solution for sites that do not want to pay for redundant WAN connections.

A simplified example of a design reflecting these ideas is shown in the illustration.

On-Site Manpower

Remote access may allow diagnosis of problems, but what happens when repair requires hands-on assistance? Typical solutions involve expensive travel by home-office support personnel, or the involvement of remote personnel who are generally unfamiliar with the broken equipment and repair procedures.

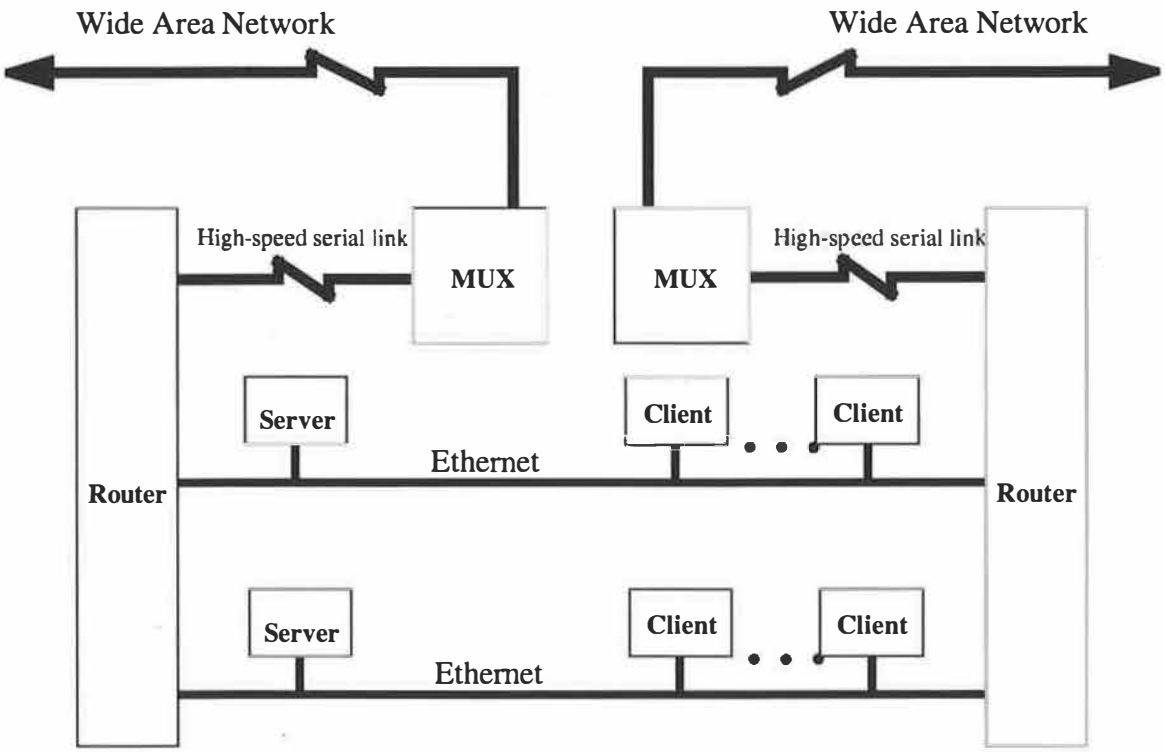
The need for on-site manpower can be reduced or eliminated through a careful combination of advance planning, negotiation of support agreements and advanced systems software.

Systems Software

Where possible, intelligent systems software should be used to assist recovery from common failures. Here are some examples:

Software can be used to perform remote unattended installation of new workstations or re-configuration of existing workstations, without requiring physical presence of a system administrator. Using this methodology, when a remote client disk fails, or a remote workstation needs to be installed, a pre-arranged third party provider delivers the equipment and plugs it into the network. From there, the configuration is completed remotely. By eliminating the need to have a system administrator present for each installation, a company can save a veritable fortune. These savings (in travel cost, man-hours, etc.), help keep the cost of remote administration at a minimum.

An Example of a High-Level Design for a Redundant Branch Office Network



Another benefit of remote unattended installation is rapid turnaround time. Typical support contracts can guarantee replacement of defective hardware within an hour or two. Once the hardware replacement is completed, the workstation can be fully configured and ready-to-use in about 20 minutes. The result is impressive: complete recovery of a defective workstation (perhaps thousands of miles away) at a minimal cost and effort.

Another example of software that facilitates remote administration is the "remote console" patch, developed by Sun Microsystems' consulting group. Use of this kernel patch allows SunOS 4.1.x clients to have remote consoles as well as active monitors. Having remote workstation consoles available allows a system administrators to fix complex problems without user intervention. For instance, suppose a system administrator added some lines to a workstation startup file (such as **rc.local**) but made a typing error. Without a remote console, the user would probably have to be talked through an ugly editing session. With remote console the SA simply halts the machine, boots it single-user and fixes the problem himself.

Support Agreements

In cases where manual intervention is required, it can often be handled by third-party support vendors. Most hardware vendors offer widespread support on either a pre-arranged fee schedule or a time-and-materials basis. This level of support is typically used for simple hardware replacement, leaving the more sophisticated tasks to be completed using remote administration methods.

User Perception

Back at the home office, end-users are often familiar with many of the support personnel as well as the structure of the support organizations. For remote users, the proper support channels may not be as clearly perceived. In robust environments, a remote user may run multiple applications supported by different

organizations. This situation, while usually manageable for a local user, can be confusing and frustrating to the remote user, who ends up shuffled from one group to another in an attempt to find the responsible party.

This can be addressed by creating a central point of contact at the home office for remote questions and problems. In this manner, the remote user always knows whom to call when a problem arises. The contact point can be as simple as a hotline phone number (with voice messaging, if unattended) or as sophisticated as a full-blown help desk. In either case, the problem is logged and the appropriate support group is contacted by the home-office contact - all transparent to the remote user, who can become comfortable with the consistent, established procedure.

System and Network Monitoring

Proper management of a company's wide-area network (the connection between the home office and the remote site) can be achieved through intelligent system configuration and careful monitoring of WAN bandwidth usage. These long-distance connections are expensive (as compared to the cost of local networks), placing a premium on bandwidth. Network-intensive applications such as NFS may provide convenient systems solutions at times, but are poorly suited for use across a WAN. One should also investigate how to allocate the necessary bandwidth in order to optimize both performance and redundancy (for example, one 256-Kb line vs. two 128-Kb lines).

Ongoing monitoring of network usage should be an active part of any system group's management responsibility, regardless of the chosen bandwidth. Examination of traffic patterns can insure network reliability by helping avoid the "12:01 syndrome" where a variety of groups schedule jobs requiring WAN bandwidth at the same time. In addition, this monitoring will help the system administrator determine the appropriate time to increase the available bandwidth.

Conclusion

The key to providing sophisticated remote administration capability is advance planning. By the time a remote system is in trouble, it's probably too late to be creative, unless the elements which enable creative solutions are in place.

When designing the remote system, ask yourself whether the home-office administrator will have a level of access comparable to that of a local machine. If not, figure out a way to achieve this. Leverage your hardware vendors to help with hands-on requirements, and use systems software to keep the amount of hands-on work to a bare minimum.

Examine how accessibility breaks down when a key component has a failure condition. Design schemes should provide redundancy in the face of individual (and sometimes multiple) failures. Use your better technical personnel in the design stage to implement system configurations that make sense in a topology which may include low-bandwidth WANs, 24-hour system usage, etc. Be careful not to blindly apply practices that work well in the home-office site without examining their applicability to a remote infrastructure.

Bear in mind that the remote users differ from local users in many ways. Provide a support structure that the remote user can become familiar with, even if she doesn't have the advantage of seeing the support personnel on a regular basis.

The right combination of planning and technology can make remote support both cost-effective and reliable. In today's distributed environment, the ability to create and manage small, remote sites can be a powerful opportunity for your company to excel where others can't. Once you've proven that you can effectively manage remote installations in Chicago, Los Angeles and Dallas, that dream of the Maui office may just be possible.

Acknowledgments

The authors gratefully acknowledge the help of Viktor Dukhovni, renowned systems management wizard, for his help with this paper.

The Operator Shell: A Means of Privilege Distribution Under Unix

Michael Neuman <mcn@lanl.gov>
Gary Christoph <ggc@lanl.gov>
Computer Operations and Assurance
Computer Security Section
Los Alamos National Laboratory

ABSTRACT

The Operator Shell (Osh) is a setuid root, security enhanced, restricted shell for providing fine-grain distribution of system privileges for a wide range of usages and requirements. Osh offers a marked improvement over other Unix privilege distribution systems in its ability to specify access to both commands and files, auditing features, and familiar interface. This paper describes the design, features, security considerations, internals, and applications of the Operator Shell.

I. Introduction

In large computing environments, support personnel have more specific responsibilities than provided for by the Unix "all-or-nothing" type of privilege system. A simple solution is to give everyone who requires system privileges the root password. For some large systems at Los Alamos, over fifty people would require system privileges: Operators are responsible for shutting down the machines, syncing filesystems in emergencies, and watching status monitors; Consultants, in the course of answering user questions, typically need to view a user's directory and files without forcing him to change access permissions; Security Administrators are responsible for auditing, monitoring, and maintaining the password database; finally, a few System administrators need complete access to the system. With potentially up to fifty people holding the root password, there are glaring problems of accountability (who did what when), administration (root password distribution--how to notify fifty people quickly and easily, and worse, having to change the password whenever someone no longer requires privileges), and user restriction (e.g., ensuring consultants aren't building new kernels).

The Operator Shell (Osh) offers a solution to each of these problems. Osh is a setuid root, security enhanced, restricted shell. It allows the administrator to carefully limit the access of special commands and files to the users whose duties require their use, while at the same time automatically maintaining audit records. The configuration file for Osh contains an administrator defined access profile for each authorized user or group. This profile lists the commands which may be run and specific access rights for files and directories. In addition to this fine grain distribution of privilege, all typed commands are logged along with a notation of their success or failure, offering a comprehensive audit log.

To the user, the Operator Shell looks like a standard C shell. It supports pipes, wildcards, aliasing, redirection, and environment referencing, except there is no longer any concept of a path. The user is restricted to only the specific commands, and specific path to those commands, that the administrator chooses. To improve functionality, some otherwise dangerous commands (e.g. *more* and *vi*) have been rewritten to support the Osh model of security. Osh does not interfere with file accesses a user would normally be able to

perform. Additionally, it allows the user access to files specified in the user's profile in the Osh configuration. For example, Osh can be configured to allow a user write access to a directory but to none of the files in it. Optionally, a file is also readable if the file owner places a key in his home directory, granting permission to a consultant, for example, to read his files. This gives a user specifically grantable control over his privacy.

II. Design of the Operator Shell

The Problem

The Operator Shell is designed to address the problem of special needs users. These users need certain system-level privileges, while being carefully limited to only the tasks they are required to perform. The needs of three types of users were considered in the design of the Operator Shell: consultants, operators, and security administrators.

The consulting group at LANL consists of seven cleared and uncleared lab employees. Hundreds of customers call in daily with questions about utilities, errors in their source code, and problems with logging in. As a result, consultants frequently need to look at the caller's files to determine the problem and offer solutions. Since they have no need for general system administration privileges, it was decided not to give them the root passwords, and force them to find a system administrator who could look through the files with them. This cost the consultants hours in tracking down a sysadmin who was willing to spend time watching a consultant use his terminal and phone. The time used on these types of problems was in the hours or days. They needed a way to look at any user's files, given that user's permission, without being able to read or write to anything they shouldn't. In addition, consultants are typically "power users" with little time, consequently the interface must be simple to learn and not force any unreasonable restrictions--they should be able to do everything under Osh that they could as a regular user.

The second type of special users are the operators. There are approximately twenty-five operators at LANL who are responsible for 24 hour computer support. They watch status monitors, reboot systems, sync disks, reset queues, etc. The operators perform all of their tasks using a procedure manual containing step by step instructions for each of the situations they're required to handle. Therefore, the interface to the Osh cannot change from the standard shell the manuals were written for, and the allowed command set and file permissions are well defined. The name and activities of the operators needed to be logged as well to provide an audit trail for system administrators to refer to if any problems occurred. With this log they can determine exactly what happened, when, and who responded. It was important to meet the needs of the operators since they comprised the majority of root password holders.

Finally, the needs of security administrators were considered in the design of Osh. Primarily, they need to edit password files and review security logs, with occasional review of user files (without the user's explicit permission) and in some cases, unrestricted access to the whole system. In addition to those user-level features, security administrators needed complete audit logs from Osh to detect abuse as well as the ability to carefully configure user and group access.

The driving force behind the Operator Shell is the difficulty of proper root password administration. Giving too many users the root password creates huge security risks, yet not giving away the root password to those users that need the privileges hurts productivity. Assuming a proper balance between security and productivity has been reached, other administration problems surface when one of the users holding the root password no longer requires it. Instead of changing and redistributing the password to all those who require it, an obviously difficult practice, the user retains root privileges until the next periodic root password change. The Operator Shell was designed to solve the problems of root password administration by addressing the special needs users who require limited system privileges.

Design Goals of Osh

Design goals for the Operator Shell were developed from the profiles of the special needs users as well as from a few general policy decisions. Osh is fundamentally designed to allow a drastic reduction in the number of holders of the root password without reducing productivity. Following are the goals which shaped the Operator Shell.

- Configuration should be easy
- Configuration should be flexible and specific enough to control all aspects of user access
- Security should be superior to standard root shells by offering thorough audit logs containing transaction time, command, and outcome
- Considering the nature of Osh, it should be extremely resistant to abuse and attack
- Resistance to abuse should not come at the cost of functionality
- Keep the code simple enough to be easily understood by administrators and system programers
- Interface should be well known to the user
- Support as many features of that interface as possible so Osh is virtually transparent
- Provide the user the same access he would have outside of the shell

III. Implementation

User Features

The interface to the Operator Shell looks like the standard Cshell with only a small number of interactive features unimplemented. Included are pipes, file redirection, aliasing, environment referencing, and wildcards. History, substitution, and job control may be added later. Integrated into the shell is a built-in help command which lists the defines the program was compiled with as well as available commands. Following is a sample session with the Operator Shell.

```
rho% uname -a
sn1054 r 6.1 rdm.56 CRAY Y-MP
rho% ls -l /usr/local/etc/osh
-rwsr-xr-x  1 root      519576 Dec 14 08:50 /usr/local/etc/osh*
rho% /usr/local/etc/osh
neuman michael (mcn)
Operator Shell version 1.2
rho,/u0/mcn #> help
Operator Shell (osh) Version 1.2
  by Michael Neuman <mcn@lanl.gov> 12/93


Defines:
NO COMPILE_TABLE
LOGGING
CHECK_ACCESS
OPER_OVERRIDE

Commands accessible:
help  cd    more  alias  wc    ls    printenv      telnet
rlogin rm   cp    w      cat
```

```
rho,/u0/mcn #> ls $HOME
osh      tcsh     uudecode    uue      uuencode    yes      yes.c
rho,/u0/mcn #> alias
Current list of aliases:
rho,/u0/mcn #> alias ls ls -CF
Alias[0]: ls  ->  ls -CF
rho,/u0/mcn #> ls $HOME
osh/     tcsh*    uudecode*  uue/     uuencode*  yes*     yes.c
rho,/u0/mcn #> rm /etc/rc
rm: Permission denied
rho,/u0/mcn #> ^D
```

Administrator Features

To the administrator, the Operator Shell is easy to configure and extremely flexible. It is based on a command matrix which can either be compiled into the shell or held in a file. If the commands are compiled in they are global for every executor of Osh and file access control lists aren't allowed. This is primarily useful if your configuration requires several copies of the Operator Shell protected by mandatory access control. The command matrix contains a list of all valid users and groups and the commands they are allowed to execute. The submatricies are additive, so for example, the special global submatrix "ALL" could specify basic commands like *ls, cd*, and *help* which everyone should be able to run, "consultants" could specify commands which should be accessible by everyone in the *consultants* group, and "mcn" could specify commands which should only be accessible by user *mcn*. If *mcn* was a member of the *consultants* group, he would have the privileges of all three of those submatricies. If "ALL" is not specified, a user can only run Osh if his username or groupname is specified in the command matrix. Following is an example command matrix.

```
ALL
{
help NULL
cd NULL
more NULL
alias NULL
wc /usr/ucb/wc
ls /bin/ls
printenv /usr/ucb/printenv
telnet /usr/ucb/telnet
}
consultants
{
finger /usr/ucb/finger
}
mcn
{
rm /usr/bin/rm
cp /usr/bin/cp
w /usr/ucb/w
cat /usr/bin/cat
}
```

In each submatrix (delineated by the curly brackets), the first column specified the command name which the Osh user would type, and the second column specified the path to that command. If it's a built-in com-

mand, it's path is specified as NULL to indicate that no external program will be run. The built-in commands are *more, cd, help,* and *alias*. Command handlers can also be implemented by the administrator to allow for specific access rights checking before calling the program specified in the command table. Currently, the Operator Shell has handlers implemented for *cp, rm,* and *vi* to check writeability of command line specified files, and *ldcache* (a Unicos command to administer logical device caches) to ensure that no command line options are passed to it. Handlers can be easily added by the administrator.

The Operator Shell's more basic functions can be configured by editing the file "config.h". This file includes the defines which specify where the command matrix is, where the logfile should go, and other basic configuration options as shown below.

Table 1: Configuration Options

Option	Type	Action
COMPILE_TABLE	boolean	Build the command table into Osh
TABLE_NAME	path	Path to the command table if not compiled in
LOGGING	boolean	Turn logging on
LOGFILE	path	Path to the logfile if logging is turned on
CHECK_ACCESS	boolean	Check for key in file owner's directory
ACCESS_FILE	string	Filename for the key
OPER_OVERRIDE	boolean	Allow users in OPER_GROUP to override CHECK_ACCESS
OPER_GROUP	string	Name of group allowed OPER_OVERRIDE status

Security Features

The purpose of the Operator Shell is to improve security of a system and several features have been added to that end. The most important security consideration is that the Operator Shell is a setuid root shell, and consequently so are all of it's children. This means that once a program is executed by Osh, the system's security rests solely in that program until it returns. Several significantly important utilities rely completely on the mandatory access controls of Unix. For this reason, *more* and *vi* have been rewritten to remove their reliance upon Unix protections. The standard Berkeley *more* allows the user to execute shell escapes and pipes as well as enter into *vi*. It has been replaced with an internal version of *more* using the curses library. *Vi* allows the user shell escapes and pipes as well as the ability to read and write to any file. *Vi* has been replaced with the publicly available *elvis* modified to disallow filename changes so that the user may only read and write to the file specified on the command line. Internal to Osh, a handler has been added to perform file permission checks on the command line filename.

File read permission is granted only if access to that file is not specifically denied and if one of the following is true:

- The user could normally perform the read
- CHECK_ACCESS is on and the file owner has the key in his home directory
- OPER_OVERRIDE is on and the user is in the OPER_GROUP
- Read permission to the file is specified in the user's access control list
- Read permission to the directory the file is in is specified and read access to that specific

file is not denied.
File write permission is granted if the write is not specifically denied, if the user could normally perform the write, or if write permission to the directory the destination file is in is specified and write access to that specific file is not denied. These access control lists are specified as part of the command submatrix. For example, the following submatrix would allow the user *mcn* read and write access to the entire /etc directory except write access to the message of the day and read access to the shadow password file.

```
mcn
{
rm /usr/bin/rm
cp /usr/bin/cp
w /usr/ucb/w
cat /usr/bin/cat
+w/etc
-w/etc/motd
-r/etc/shadow
}
```

In addition to these access control mechanisms, thorough audit logs are kept. All typed commands are saved in the logfile with real user ID, date and time, command typed, and an indication of success (+) or failure (-). Following is a sample from the log where mcn is a user allowed to modify the kernel, but not anything else.

```
LOGIN: mcn ran osh at Tue Feb 15 18:54:46 1994
mcn (2/15/94 18:54:47): help      +
mcn (2/15/94 18:54:51): ls /etc      +
mcn (2/15/94 18:54:56): rm /etc/fstab      -
mcn (2/15/94 18:55:03): ls /      +
mcn (2/15/94 18:55:07): rm /vmunix.old      +
mcn (2/15/94 18:55:11): ls -l /      +
mcn (2/15/94 18:55:27): mv /vmunix /vmunix.old      +
mcn (2/15/94 18:55:47): ls -l /vmunix.old      +
mcn (2/15/94 18:55:55): cd /var/log      +
mcn (2/15/94 18:55:56): ls      +
mcn (2/15/94 18:56:04): rm syslog.7      -
mcn (2/15/94 18:56:06): cd /var/adm      +
mcn (2/15/94 18:56:07): ls      +
mcn (2/15/94 18:56:23): rm lastlog      -
logout: mcn left osh at Tue Feb 15 18:56:33 1994
```

In addition to access control lists and logging, security is enhanced in other, more subtle ways. By setting both the file and the explicit path to that file, it is assured that the commands are run from their proper location. This avoids many of the more common setuid program exploitations such as IFS and PATH environment variable modification. A useful side effect of the Osh program being setuid root is the command matrix file can be set with read permission to no one. Consequently, malicious users aren't given any hints as to whom has the most power and is therefore worth the most time to hack.

Internals

There are two primary components to the Operator Shell's internal security. The first is the means of executing external programs and the second is the file access protection mechanisms. The method of executing external programs has been carefully constructed to avoid many of the security holes involved with setuid programs executing other programs. The file access protection mechanisms are designed to allow the administrator to configure the exact permissions for file accesses and allow the consultant access to users' files, all without restricting the user beyond his normal capabilities.

When a the user types a command, it's entry is looked up in the command table. If the entry exists and has an associated path, it's corresponding handler is called (if there is one) which checks for command line file writeability, then the function execute() is called which determines if the program is a shell script or an executable, then calls execv(3) with the explicit path and options. The reason for determining if the file is a shell script or not is a large number of shell scripts do not have a "#!" header and execv(3) cannot handle these files directly, so the execute() function inserts the Bourne shell as the program to be executed. If the entry is internal, the command's handler is called directly.

If the command can write to files, or if the command line contains a redirect to a file, writeability is allowed if any of the following rules are true:
- The file is specified as +w in the user's access control list
- The parent is specified as +w, and the file is NOT specified as -w
- The file would normally be writable by the user and the file is NOT specified as -w

The commands which need to check writeability must provide handlers to perform the access checking internally. This is obviously necessary for commands like *cp* which require the first argument to be checked for readability and the second for writeability.

For every command, the default security checking routine will test each argument on the command line for file readability. If the argument doesn't exist as a file, then it is assumed the argument is an option and is ignored. For this reason, handlers must be created if argument writeability must be checked.

The only functions a handler needs to perform is a test of the validity of the command line options, a notation of the command's success or failure, and then a call to execute(). Handlers can be used to check writeability of an argument, force an option to a program, ensure arguments weren't sent to a function, etc. For writing handlers, the following routines are important.

Table 2: Support Functions

Function	Arguments	Return codes
acl()	char *filename, char mode	+1 if filename is listed as a + for the given mode, -1 if filename is listed as a - for the given mode, and 0 if the filename is not listed in the acl
check_access()	int argc, char **argv	+1 if all arguments are readable (or don't exist), 0 if one argument is not readable
writeable()	char *filename	+1 if filename is writable, 0 if not writable
execute()	int argc, char **argv	Executes the command line, should not return except on error with execv(3)

The check_access() routines iterates through all command line arguments before calling any handlers. It does this according to the following pseudo-code:

```
FOR i=0 TO number of arguments
   IF argument(i) +r in acl THEN
     NEXT i  { File is readable }
   ELSE
     IF argument(i) -r in acl THEN
       invalidate command line
       return from routine
     ENDIF
   ENDIF
   {if we get this far, that means the argument isn't in the acl}
   IF argument(i) exists as a file THEN
     IF real uid is allowed to read argument(i) THEN
       NEXT i  { File is readable }
     ELSE
       IF (not CHECK_ACCESS) or  ("ACCESS_FILE" exists in argument(i)
            owner's home directory) THEN
         NEXT i  { File is readable }
       ELSE
         IF (not OPER_OVERRIDE) or (user not a member of OPER_GROUP)
          invalidate command line
          return from routine
         ENDIF
       ENDIF
    ENDIF
   ENDIF
NEXT i
```

Handlers are added by the following procedure:
1) Add the handler function prototype to the prototype section in struct.h
2) Add the command name and handler function name to the Internal[] array in struct.h
3) Modify the NUMINT define right below the Internal[] array to reflect the number of internal handlers
4) Add the handler code to handlers.c
5) Recompile

IV. Summary

Osh is best applied to any situation where specific privileges need to be given to specific users. It allows the administrator to specify access to both commands and files for individual users and groups, automatically maintains audit logs, and is easily configurable and modifiable for any situation. These features make the Operator Shell superior to root password distribution or any other privilege distribution system. At LANL, the Operator Shell has cut the number of users who require the root password on our supercomputers from more than 50 to less than 10. The security and administration benefit of this is remarkable.

Comparison with Other Systems

Similar systems to the Operator Shell are Sudo (by Jeff Nieusma) and Runas (by Christopher Carpinello).

The interface to both Sudo and Runas is via the command line. For example, a user would type "sudo vipw" which, if the access control file allows it, will run vipw as root. This is a quicker interface than Osh for short administration tasks, but it doesn't allow the administrator to protect commands from dangerous options or specify access controls to files.

Table 3: Comparison of Osh, Sudo, and Runas

Feature	Osh	Sudo	Runas
Interface	Complete shell	Command line	Command line
Access Control	Programs and Files	Programs ONLY	Programs ONLY
Commands run as	root ONLY	root ONLY	Any user
Command protection	Protection against any options with handlers	NONE	NONE
Logging	Command and it's success or failure	Attempts to run Sudo	Attempts to run Runas

Availability

The Operator Shell is available in two ways:

Email: send mail to Mike Neuman <mcn@lanl.gov>
FTP: Anonymous FTP at ftp.c3.lanl.gov[128.165.21.64]:/pub/mcn/osh.tar.Z

Osh is known to run under Unicos 6.1 and 7.x, SunOS 4.1.3, and Ultrix 4.2.

A New Network for the Cost of One SCSI Cable:
A Simple Caching Strategy for Third-Party Applications

Hal Pomeranz
QMS Inc., Santa Clara

The network at QMS Santa Clara was becoming overloaded by NFS traffic, particularly during large builds. To help alleviate this problem, software was developed to automatically cache frequently used application binaries and support files. This paper discusses implementation concerns and trade-offs made during the development of this software. The result of the software's deployment was increased network throughput and more predictable network response.

1.0 Introduction

Ten megabit ethernet has been the LAN standard for over a decade. Faster standards have become available relatively recently, but it is still not cost effective to deploy, say, FDDI to every desktop. While networks have been stalled at ten megabits of shared bandwidth, the workstation peers on those networks have been doubling in speed every 18-24 months. With this kind of mathematics, it's easy to see why many networks today are reaching saturation point.

At QMS Santa Clara, we have what is kindly referred to as a *legacy network*. In other words, we have a four year old strand of thick coax which currently supports all of our networked devices with no bridging or routing. While this was adequate given the number and power of the machines at the time the network was installed, we have added many more and faster machines, as well as connecting to various remote locations and channeling all corporate Internet traffic through our site. Clearly we need a new network, but just as clearly we cannot afford to spend the money on it right now.

Organization of our computing devices is fairly standard for a commercial software development environment. Our desktop machines are configured in datafull mode: each desktop machine has a local (identical) copy of the operating system, a local swap area, and 50-60 megabytes of space for user home directories. Development tools are maintained under a `/usr/local` filesystem on our file servers and exported via NFS to the desktop machines. The file servers also hold our master build directories and project development storage.

We are also great believers in parallel build technology. Idle machines on our network are available as a compute farm to enhance the performance of system builds. While most environments can rely on their desktop machines being idle as much as 90% of the time, our machines are used more or less consistently throughout extended business hours from 9am to 8pm.

Currently, our network supports two file servers and approximately 40 desktop workstations. Any one of these machines is capable of using all or nearly all of 10 megabits of network bandwidth on its own. Users were beginning to complain that they were being bottlenecked by the network. During large builds, the NFS resources of the file servers would be overwhelmed by having to serve both the source code files and the executables used to compile that source. All other work would essentially stop while the build was in progress, or the build would abort with an NFS error.

2.0 Finer Focus

One indicator of network load is the collision rate of systems on your network. On a broadcast network such as ethernet, a collision occurs when two devices attempt to transmit at approximately the same time. When

this happens, both devices are obligated to back off for some random period of time and then retransmit. The collision rate is the per machine ratio of collisions to total number of packets transmitted by your machine (both of these values can be collected on most UNIX system using the `netstat -i` command).

Probabalistically speaking, collisions must occur occasionally. As the number of machines and the amount of traffic they generate on your network increases, so does the frequency of collisions. A collision rate of 5% generally indicates that it is time to start thinking about subnetting or bridging your network. Users will begin to notice significant performance degradation as the collision rate moves into double digits. Before we deployed parallel build software at QMS, most of our desktop machines had collision rates in the 5-10% range while our two file servers were in the 10-20% range. After we started doing parallel builds, the collision rates doubled across the board. Peak collision rates above 40% (one hour average) have been reported on the file servers.

Analysis of our network traffic showed that the largest single contributor to network load (both in terms of number of packets and total amount of data) was NFS. The NFS traffic was essentially divided into three groups:

- read-only requests for development tools and supporting files from `/usr/local`
- read-write requests to system build directories
- read-write requests to user home directories

Of these three categories, greater than 60% of the traffic was due to the first category, and the remainder of the traffic was more or less evenly divided between the other two categories.

To understand this distribution, consider what happens during a system build. For each C source file, first invoke the pre-processor (NFS read to grab the executable), read the source file and any include files (NFS reads), invoke the compiler (another read to get the executable), invoke the optimizer (ditto), and write back the object file. The total size of all executables used during this procedure is often an order of magnitude more data than the combined source and include files and the object that they produce. When the object files are linked to form an executable or a library, there is a significant amount of NFS read traffic in the build directory and typically a number of sizeable write requests, but large links occur relatively infrequently during the course of the build.

In our environment, developers work on components in their home directory, building, testing, and debugging before release. This generates a more or less consistent level of traffic through user home directories. Our system integrators periodically combine components and produce full system builds which generate large traffic peaks in the system build hierarchies. Averaged over time, this produces a more or less even amount of NFS traffic in each of these two areas.

To understand the phenomenon of our file servers "locking up" due to overwhelming demands on their NFS resources during system builds, we conducted a small experiment. We made a copy of one of our compiler hierarchies on one of our desktop machines and ran a system build using this copy of the compilers rather than the file server copy. Source code files were still obtained from the file server. As expected, NFS load on the file server was reduced by nearly 60%. We were also happy to note that processing was able to continue for other users, though slowdowns were noticed.

3.0 Solutions

One solution to avoid having to get our development tools across the network is to have them locally on each client machine. Unfortunately, our `/usr/local` area occupies some 450 megabytes of space, and if we cannot afford to upgrade our network, we certainly cannot afford to buy 500 megabytes of additional disk for all of our machines (to say nothing of the administrative nightmare of trying to keep all those disks in synch).

Fortunately, a 90/10 rule applies to this problem. Paul Anderson[1] has demonstrated that 90% of this kind of traffic can be saved by using an amount of disk space equal to approximately 10% of the total size of the partition as local cache. The cache contains the most frequently used files from the `/usr/local` area, determined on a per-machine basis. To obtain this cache space on my desktop machines, I was faced with the choice of doing away with either the local partition for `/usr` or for home directories. For primarily administrative reasons, I chose to do away with the local home directories. With no user data on the desktop machines, they essentially became interchangeable units. Disaster recovery was simplified because I could simply swap in a preconfigured spare in place of any failed drive. Backups could be reduced or eliminated on the desktop machines because the data they contained was essentially static.

Paul had developed some modifications for his `lfu`[2] tool to implement such a caching strategy. `lfu` was originally developed to solve the problem of keeping entire software directories in synch across multiple file systems and contained a significant amount of machinery towards this end. I decided that the tool was "too big a gun" for my application, since I was really only interested in the caching portions. It would be easier, and more maintainable in the future, to develop my own tool to implement Paul's ideas.

Another alternative open to me was to deploy AFS, which has a built-in caching mechanism. The biggest strike against AFS deployment was that it cost money, in the $10K range for a site my size. We were definitely in a "spend no money" mode. Obviously, AFS requires kernel modifications, and while it has become relatively easy to build custom kernels, tieing yourself to a third party for kernel code means that you tie yourself to their upgrade schedule, which may lag months behind your vendor's upgrade schedule. AFS also tends to require more cache than I had available, though on the plus side I would have been caching more than just the development tools under my `/usr/local` area.

Another significant barrier to AFS deployment is the barrier of "guruhood". That is, I would have had to immerse myself (and my users to some extent) in a radically different paradigm for sharing files. While I tend to be an aggressive implementor of new technology, I was unwilling to face the pain at that time. Also, I am a solo administrator at my site and I worried about my successor having to drop into a significantly new environment, or tieing the search for my replacement to somebody with AFS experience.

4.0 Software Overview

The basic idea of Paul's caching strategy is elegantly simple. Mount `/usr/local` from the file server onto some other (hidden) directory on the desktop machine. In the cache area on the desktop machine's local disk drive, create a link tree which mirrors the `/usr/local` filesystem from the server: the directory structure is preserved, but all files are replaced with symbolic links. Since all files appear to be in the correct place, all software tools function normally. Now scan the links periodically and see which ones are being accessed most frequently. Replace these links with actual copies of the files they point to, subject to the limitations of the size of your cache.

4.1 Synchronization

The high level overview presented above glosses over some difficult issues. The first of these is the issue of keeping the cache areas in synch in the face of changes to the directory structure or updated versions of files appearing in the "master" `/usr/local` directory on the file server.

1. Anderson, Paul, "Effective Use of Local Workstation Disks in an NFS Network", *USENIX LISA VI Conference Proceedings*, 1992.
2. Anderson, Paul, "Managing Program Binaries in a Heterogenous UNIX Network", *LISA V Conference Proceedings*, 1991.

If new files are created in directories on the master server, new links must be created in the corresponding directories in the cache area. New directory structures added on the master server must be duplicated recursively in the cache area. Both of these duties have been integrated into the same script which implements the rest of the caching strategy, since it is wasteful to traverse the filesystem once to synchronize it with the master server, and then again to collect frequency counts and cache files. The routines used to create new directory structures in the cache area can also be used to build the initial link tree against the master server.

The software must also detect if a file has already been cached, but the file has been updated on the master server. Currently, the *mtime* of the cached file is compared against the *mtime* of the file on the master server, and the cached file is updated if the master server's is more recent. This implies that updating *mtimes* on the master server using a command like `touch` will force cache updates even though the contents of the file have not actually changed. The alternative, however, is to do a byte-by-byte comparison of every cached file against the master, which is costly both in terms of computation and I/O.

4.2 Moving Files Into/Out of Cache

Due care must be taken when moving files into or out of the cache. An application file may be open, and simply unlinking the file and replacing it with an updated file or symbolic link is a good way to cause the program to dump core.

The proper way to move files in and out of cache is to employ the *rename(2)* system call. *rename* will update the file system links to point to the new data, but will not free the space held by the old data until all references to that data are closed. The process for moving a file into the cache, then, is to copy the file from the master server to some temporary location in the same filesystem and then call *rename* to move the temporary file into place.

After copying a library file in this fashion, it is necessary to re-run `ranlib` to update the archive's table of contents (at least if you wish to avoid annoying messages during system builds).

4.3 Link Information Can Mislead

To determine which files are being accessed most frequently, the caching software compares the *atime* (last access time) of the symbolic link or cached file against the last time the caching software was run. There are a number of problems with this strategy.

First, the data acquired is rather granular. All that is known is that the file was accessed at least once since the caching software last visited. Obviously, it would be preferable to know exactly how many times a given file was opened, but perfect information would essentially require intercepting all such open calls, possibly at the kernel level. It turns out that even a very coarse granularity is sufficient.

A second problem is that some applications such as `emacs` or `xdm` tend to run more or less continuously. The "number of times accessed" metric is not appropriate in these cases. On possibility would be to interrogate the kernel to find out which files are open at the time the caching software is run and consider these to be "accesses". This does not turn out to be very portable, however, since it is inherently kernel specific. Instead, the caching software reads a data file created by the administrator on the desktop machine which lists those files under the master `/usr/local` area which must be forced into the cache. This file can be configured on a per machine basis, but it is administratively easier to maintain a single file which can be distributed to all machines.

A third problem with looking at atimes on symbolic links is that certain forms of the `ls` command (e.g. `ls -lF`) can update the *atimes* on all links in one directory or several directories at once. The caching software combats this phenomenon by keeping track of the oldest *atime* reported for any symbolic link in a given directory. If this value is newer than the last time the caching software was run, assume that all *atimes* were updated by an `ls` at this time and only count as accessed those links whose *atimes* are greater than this

value. Actually, it is necessary to add an additional few seconds as a "fudge factor" since the `ls` command in large directories (e.g. X Windows font directories) can actually take a significant amount of time.

4.4 Limiting the Cache

The first time the caching software is run, the cache may become flooded with files that are not actually used frequently. To avoid this problem, files are only cached if they have been accessed some minimum number of times. Five accesses appears to be an acceptable limit.

The administrator may wish to never cache some classes of files. Files in this category may include manual pages or other documentation, lock files, or frequently updated data like "high score" files for games. The caching software does consult a per machine configurable file for the names of files or directories the should never be injected into the cache.

It may be undesirable to completely fill the disk space available in the cache partition. Perhaps the cache shares a filesystem with other data, or perhaps the administrator simply wants a safety valve against unforeseen difficulties. The caching software has a "minfree" parameter which can be used to specify how much disk space in the partition cannot be used for cache. The partition will always have between 100 and 110% of this amount of space available.

5.0 Implementation

Implementation (in Perl) of the initial version of caching software took approximately two weeks, but less than forty man-hours. There followed another two weeks of testing on a limited number of machines and associated bugfixing. Additional improvements were made to the caching software as a result of feedback I received from a "Works in Progress" talk at LISA VII. The resulting Perl script is less than 500 lines of code, including comments (about 16K in size).

Once the script was ready, I needed to find space on my file servers to migrate two gigabytes of home directories into. Luckily, I had been forced to send a one gigabyte drive to our Boulder office for an emergency need, so I was able to justify a replacement. I had been planning to use the drive to increase the size of my news spool, but that obviously was not going to happen. One of our desktop machines had an external one gigabyte drive attached, which had been used previously for a now-defunct project. It currently contained, let us say, information that did not pertain to the business mission of QMS. I swapped the user for a smaller drive and some space on our optical jukebox.

Management was more than willing to support my proposal because it only "cost" the company a portion of my time and resources. The "cost of one SCSI" cable angle comes in because I did not have the right cable for the SCSI interface on my file server. Having solved the disk space and management approval issues, I then spent a weekend installing disks, migrating home directories, and initializing cache areas. After two weeks, the cache areas had essentially stabilized.

Our `/usr/local` area contains approximately 18,000 files occupying some 450 megabytes of disk space. The link tree in the cache area occupies about 15 megabytes of space with no cached files. Our cache areas have stabilized with an average total disk utilization (including links) of about 50 megabytes. The caching script takes less than 3.5 minutes to run on a Sparc IPC with 12 megabytes of RAM, and occupies less than 2 megabytes of RAM while executing. It is possible to run the script while a user is on the machine without significant performance impact.

Collision rates dropped approximately 5-10% across the board once the cache areas had been established. The relatively small drop in collision rates is due primarily to an overall increase in network traffic. Since the file servers were no longer exhausting their NFS resources, it was possible for more developers to conduct more parallel system builds simultaneously. Networks, like nature, abhor vacuum, and spare network capacity was quickly used.

While overall collision rates did not drop drastically, collision rates on certain machines dropped to levels comparable to the time prior to the introduction of parallel build technology. These machines were those which tended to have users more or less consistently at the console: when the console is not idle, the machine is not available for parallel builds.

Most importantly, overall network performance has become much more predictable. We no longer have sudden work stoppages when one of the file servers becomes overloaded because this condition never happens. Even with nearly forty machines involved simultaneously in a distributed system build, our file servers are capable of coping with the load.

An unexpected benefit of the caching area is that desktop machines are more robust in the face of file server crashes. When application files were kept only on the master servers, a file server crash would often result in a user's X session or other application simply aborting ungracefully, often causing considerable loss of work. Since commonly used applications are now cached locally, this rarely happens. Since other directories (home directories, component trees, mail boxes) are mounted from the file server, the users still don't get much work done at these times, but at least they don't lose what they've already done.

A negative effect of caching is that the process of installing new software or updating existing software becomes more complex and time consuming. Once the master servers have been updated, the administrator either has to trigger the caching software manually to update the cache areas, or they have to wait for the next automatic run. In our environment cache areas are updated twice per day: once during lunch hour and once at night. This has proved sufficient to date, although there have been a few occasions when the caching software was triggered manually to update all clients in a more timely fashion. It is at these times that the limited system impact of the caching software is most appreciated.

6.0 Conclusion

Overall, users and management alike are happy with the solution. Predictability of network response is most important in this case, though performance (particularly during distributed builds) has improved as well. Certainly, this is not a long term solution for my environment: network partitioning is required as soon as the capital budget becomes available.

I believe significant gains could be had by obtaining additional disk space on the desktop for user home directories. Most of our existing machines have 200 megabyte drives installed. A cost effective solution to disk space would be to upgrade half of the machines to 400 megabyte drives and install their old drives in the other half of my machines. Collision rates on the few machines I do have with 400 megabyte drives are several percent lower than on similar machines equipped with 200 megabyte drives.

7.0 Availability

The following files are available via anonymous FTP from `ftp.qms.com` (`161.33.3.1`) under the directory `/pub/cache`:

`nightly`	Perl script to implement caching strategy
`nightly.man`	Manual page for same
`dont.cache`	Example exclusions file
`must.cache`	Example list of files to force into cache
`nightly.shar`	A shell archive containing all of the above

Guarding The Fortress

Efficient Methods To Monitor Security on 300 Systems

Michele D. Crabb (crabb@nas.nasa.gov)
- Sterling Software / NASA Ames Research Center

ABSTRACT

With the increased proliferation of desktop UNIX workstations, computer security has become an increasingly larger headache for sites which have several hundred to over 1000 systems. Most people would agree that keeping a watchful eye on a handful of workstations or mainframes is a simple task. However, keeping a watchful eye on several hundred workstations of differing architectures can be a security administrator's never-ending nightmare, if not done efficiently. Unlike many other types of UNIX system administration tasks, which can be placed on the queue and done at a later date, delaying the completion of a security task, such as the installation of a security patch, could leave a site very vulnerable to an intruder attack. Therefore, in order to adequately monitor and protect a large number of systems and users, security tasks must be well organized, efficient, and automated, if at all possible.

This paper will present an overview of how system security is currently maintained on the over 300 systems at the Numerical Aerodynamic Simulation (NAS) facility at NASA Ames Research Center. The discussion is divided into five areas: why security is needed; what policies and procedures are used; what types of security awareness training is provided; the security monitoring and checking tools that are used, and some future plans for security at the NAS facility.

Introduction

Computer security has been a growing concern in the government and private sectors since the Morris Worm Experiment in 1988. That single incident gave birth to many jobs for UNIX security people, including myself. The UNIX security arena has been growing at a steady pace ever since. UNIX system security in the large heterogeneous environments of the 90s seems to be a security administrator's never-ending nightmare. UNIX system security is no longer just ensuring that all accounts have passwords and restricting who has the *root* password. Instead, system security is now a combination of good system administration, well-defined security policies and procedures, and regular security training programs for both the system support personnel and the user community. Neglecting any one of these three areas can create a weak link. Due to the importance of many security tasks, such as monitoring log files and installing patched versions of binaries, they cannot be delayed like many other system administration tasks. Therefore, in order to adequately monitor and protect a large number of systems and users, security tasks must be well organized, efficient, and if at all possible automated.

A prerequisite to gaining efficiency is the knowledge and understanding of the task at hand. In the last several years, there has been a large number of books and papers published on UNIX security, and how to improve security at a site. There has also been a large amount of security related software that has been published on the Internet. With the use of these tools, anyone can effectively monitor and manage security in a large, heterogeneous environment.

The NAS environment is comprised of over 300 computer systems running some flavor of the UNIX operating system. The NAS systems are interconnected into a heterogeneous network using various types of networking hardware and are supported by over 100 personnel, who provide ongoing support and software development. These systems are of differing architectures which include Sun SPARCstations, SGI 4D workstations, SGI 4D/480 multi-user systems, Cray Y-MP C90s, Convex C-3240s and several massively parallel processing machines. The NAS facility, which primarily provides supercomputing services to other NASA sites, large aerospace corporations and a large number of universities, has over 1500 users

nationwide. The security on these systems is primarily monitored and maintained by one full time employee. The main objectives of computer security at the NAS facility are to ensure:

- Security meets all NASA headquarters and Ames guidelines and requirements.
- The support staff is capable of responding to a security incident in a timely and orderly manner.
- An adequate level of security monitoring is performed on all systems.

Prior to entering my current position as the NAS security analyst, there were no formal written policies or procedures for maintaining security at the facility. System security was seen as a side task for the system administrators. During my first two years in the position, my main focus was to develop a framework for providing security. The challenge in the task was many methods and procedures which work for a small number of systems do not scale well to several hundred systems. As I began to tackle this monestrous tasks, I saw seven major areas of security which I needed address. These areas are:

- Network access monitoring and restriction
- Files system auditing
- Password validation and management
- Special access management
- User Account management
- User awareness and training
- Policies and procedures to make it all work

The discussion focuses on how these areas are addressed at the NAS facility. First I discuss some of the motivators for providing good system security. Next is a brief discussion on the importance of a security awareness, which includes a description of the NAS program. Then I examine specific security concerns and how they are addressed. This section of the paper includes a discussion of the many security tools used at NAS perform such tasks a network access monitoring and file system administration. Finally, I describe some of the future plans for security at the NAS facility.

Strong Motivators for Good Security

There are a number of reasons why UNIX system security has continued to grow in popularity and importance over the past five years. Prior to the Morris Worm Experiment of 1988, little attention was given to the subject of UNIX security. However, that single event, which brought the Internet to a grinding halt coast to coast in under three hours, made the UNIX world stand up and take notice. Since then, UNIX security has blossomed into a recognized profession.

One of the paramount forces behind the continued growth of the field is the massive proliferation of UNIX systems throughout the nation. As larger numbers of sites acquire UNIX systems, two distinct patterns emerge. First, more and more information is made available on-line. The wealth of information on the Internet is virtually endless. Even the White House is "on-line" these days. As more and more information becomes available on-line, people want access to this information. In some cases, the desire for access to information is for illegal reasons. The second pattern which emerges is the increased popularity of UNIX has led to an established underground community of Internet intruders. These are people who view the Internet as an endless playground. They range from juvenile computer kids who intrude for fun, to professional technology embezzlers who intrude for profit. Most of the work done in the UNIX security world over the last several years has been to provide better protection for on-line information and to provide better methods to keep the Internet intruders at bay.

One of the strongest motivators for good system security at many sites is the need to protect information from disclosure, modification or destruction. The information on the systems may range from student or employee information to vendor marketing analysis and strategies. At the NAS facility, our users' data ranges from C and Fortran source programs to experimental data. While most of this information is not classified, there is a level of sensitivity and a need for protecting the information. Furthermore, NAS users

expect their data and information to be safe on our computers.

Another strong motivator for system security is the need to meet regulations and requirements from within the organization. Almost all government organizations and many private sector companies have various regulations and requirements for protecting their computing resources. At the NAS facility, we must meet various regulations and requirements from three different levels of the NASA organizational structure: NASA headquarters, NASA Ames, and our own facility. Without formalized regulations and requirements from higher levels within the organization of a facility, many security procedures and measures may never be put in place.

Finally, another motivator for providing good system security is previous experience with Internet intruders. Unfortunately, many facilities give little thought to computer security until a large-scale incident occurs. A prime example of this phenomenon was the Morris Worm Experiment. Prior to that incident, the NAS facility did not have a position dedicated to computer security. I have heard similar stories from many system administrators I have spoken with at various UNIX conferences.

Policies and Procedures Needed For Efficient Security

Policies and procedures are key elements of any successful organization. Security policies and procedures are especially important as they define the rules to live by and the penalties for breaking those rules. Security policies also help identify security threats and vulnerabilities. They will outline what behavior is, and is not allowed, on the system. Security policies may even help in prosecuting intruders or serious violators of the policies. There are some basic elements that all security policies should include. The most important element is which actions are allowed and not allowed in the local computing environment. For example, are users allowed to play electronic games? Are users allowed to run password crackers on the system? Security policies should also define what actions will be taken against people who violate the policy. For example, will a user's account be disabled because the user was running a password cracking program on the system? Security related policies should also outline the rights and responsibility of the users as well as the system administrators. All policies should be made available to all people who are effected by them.

The NAS facility has a variety of security policies already in place and several new policies in the acceptance process. The intent of all NAS policies is to ensure that computer security on NAS computing resources is maintained according to NASA regulations and policies, while at the same time not impeding the ability of NAS users and support staff to perform their work. All of the NAS computer security related policies are described below.

The first policy is the *Acceptable Use Statement for NAS Systems Division Computing Resources.* The purpose of this policy is to increase the awareness of computer security issues and to ensure all NAS users (scientific users, support personnel and management) use the NAS computing resources and facilities in an efficient, ethical and lawful manner. The policy is a collection of twelve rules. Every NAS user is required to sign this agreement as part of their account approval process. The policy is very explicit about how violations will be handled. It states "Any noncompliance with these requirements will constitute a security violation and will be reported to the management of the NAS user and the NAS DPI-CSO and will result in short-term or permanent loss of access to NAS Systems Division computing systems. Serious violations may result in civil or criminal prosecution." The full text of this policy can be found in Appendix A.

Closely related to the *Acceptable Use Statement* is the *NAS User Account Policy.* This policy outlines the requirements for accessing or requesting a NAS account. Accounts are only issued to people residing in the United States and only for approved projects. Accounts which are inactive for a period of 90 days or more will be disabled, and after an additional 30 days, the account is archived and removed. Dormant accounts can be a source of major problems if the accounts are discovered and compromised by intruders. Many sites, especially universities, will allow users to keep their accounts when they no longer have a valid reason for the account. I consider this to be a poor security practice and something that should be avoided.

At the NAS facility, accounts are archived and removed after the completion of a project.

Due to the complexity and organizational structure of the NAS facility, a large number of support personnel require special access (e.g., *root* access). The next two policies were written specifically to deal with special access. The first is the *Special Access Policy*. This policy also provides a set of requirements for the regulation and use of special access on the NAS systems. The policy provides a mechanism for the addition and removal of people from the special access database and a mechanism for periodic reviews of the special access database. As a part of the policy, users are required to sign the *Special Access Guidelines Agreement*. This agreement outlines the many dos and don'ts of using special access on NAS computers. The other policy is titled *Computer Usage Guidelines for NAS Systems Division Personnel*. This is one of the newer policies at NAS and is still in the final approval stage. The purpose of this policy is to establish usage guidelines for support staff with *root* access and is intended to protect the rights and privacy of NAS Systems Division clients as well as those of the NAS staff. The three major areas covered in the usage guidelines are: how to protect the privacy of clients data when working with them on a problem; how to deal with proprietary information that may be stored on the NAS computer systems; and when and how to perform a security investigation on a person suspected of violating policy.

The next two policies deal with system installation and configuration, rather than system usage. The first is the policy *Regulatory Use of the /etc/hosts.equiv, /etc/hosts.lpd and .rhosts Files*. The purpose of this policy is to provide a set of requirements for regulating the use of the system trust files. The policy outlines the necessary requirements for adding a host entry to the */etc/hosts.equiv* or */etc/hosts.lpd* files. The policy also discusses what type of *.rhosts* entries are allowed. The other system configuration related policy is the *NAS Computer Network Connection Policy*. This policy outlines the requirements and contraints for attaching a computer to the *nas.nasa.gov* domain. The intent of the policy is to ensure that all systems installed on the NAS network are configured and maintained at appropriate levels of security, and the security measures implemented meet NASA regulations and policies as well as any Ames regulations and policies.

In addition to the policies discussed above, there are two very important security procedures implemented at the NAS facility. The first is the *Security Incident Escalation Procedure*. This procedure describes the escalation process for various types of security incidents. The escalation process varies depending upon the severity level of the incident. For example, account sharing is considered a minor (or level 1) incident. If a case of account sharing is discovered during the off hours, it would not be a necessary to contact the security person right away. However, for such incidents as a suspected computer virus, immediate notification of the computer security analyst and other support personnel would be extremely important. The other procedure, which is closely related to the escalation procedure, is the *Security Incident Handling Procedure*. This is a document that every computing facility needs. The purpose of a *Security Incident Handling Procedure* is to provide reasonably detailed instructions on how to handle the various types of security incidents. The NAS procedure outlines the areas or responsibilities for support staff, lists some general procedures, and provides detailed instructions on how handle specific types of incidents. To simplify the response process, security incidents are classified into four areas, and each area is handled differently.

Some of the NAS policies were implemented as the result of various security problems. The *Acceptable Use Statement* was prompted by several incidents where employees were reprimanded because they were reading people's email and looking at their files. The NAS *Computer Network Connection Policy* was inspired by a major security incident at the NAS facility which involved a poorly administered workstation that was installed and maintained by a supporting vendor and not by the NAS support staff. Security policies and procedures should be written at the beginning of the life cycle of a computing facility, instead of after the fact, or as the result of a security incident. All of the policies described in this section are policies and procedures that are appropriate for any large computing facility.

Security Awareness

In addition to having a set of well-defined security policies and procedures, a successful computing facility needs to implement a security awareness program. One of the major objectives of a security awareness program is to ensure that the users, management and system administrators understand the different roles they play in the overall security of the facility, as well as the policies and procedures that pertain to them. If the users understand the security issues and the motives for implementing the different policies and procedures, they are more likely to adhere to the policies and report suspicious system activity.

A security awareness program can consist of hardcopy reading materials, on-line information, training classes or training videos. All users at a facility should be informed of, and provided access to, all security-related policies and procedures when they first begin their employment. On-line access to information is preferred over hardcopy, since hardcopy materials are usually misplaced or discarded. System support personnel should be provided training on an annual or bi-annual basis. The training provided for the support staff should include a more technical overview of security so they can play an active role in maintaining the security of the facility.

The security awareness program at the NAS facility consists of hardcopy reading materials, on-line information, training classes, and a training video. All of our security-related policies and procedures are available on-line and hardcopy to NAS users. They can view the information on-line using the *gopher* or *Mosaic* browsers. The *NAS User Guide*, a document made available to all NAS users, contains a separate section on system security at NAS. The information available in the *NAS User Guide* includes: the *NAS Account Policy*; tips on selecting and maintaining passwords; setting files permission; auditing of users home directories and environment files; using *.rhosts* files and workstation console security. All new NAS users are made aware of this information when they receive their new account notification. If the need arises to inform users of important security information, a bulletin is posted to our on-line information systems and a note is made in the system MOTD. If a limited number of users are involved, the users are contacted directly via email or a phone call. The security awareness program for the support staff also includes an annual extensive full-day tutorial on UNIX system security. Enrollment is open to all interested persons. The class is held locally at the NAS facility. All members of the NAS support staff are required to attend the class at least once. The full-day class has been video taped and is available to NAS users and NASA employees.

Methods and Tools For Security Monitoring

Over the last several years, there has been a wide variety of security-oriented software which has been made freely available on the Internet. This section discusses some of these tools, as well as some locally developed tools which aid in the security monitoring of the NAS facility. Other available tool are described briefly. The discussion is oriented around security functions and the tools which aid in the monitoring of that specific function.

Our first line of defense at the NAS facility is host-based filtering and monitoring of all network connections (e.g., *rlogin* and *telnet*). The use of a host-based filtering mechanism was prompted by a very large security incident which occurred at the NAS facility in late 1991. The NAS computer systems were the victim of repeated intruder attacks over a two month period. After the third successful break-in, a decision was made to install the *tcp_wrapper* software (i.e., *tcpd*) to provide host-based filtering. In order to reduce the number of NAS hosts which could be reached from the Internet, a filtering scheme was implemented which only allowed remote sites to connect directly to a few of our systems. The NAS workstations were configured to only accept connections from hosts within the local domain. Our main philosophy behind this scheme was to reduce the scope of the problem by disallowing remote connections to the over 300 NAS workstations.

The *tcpd* program is configured to log all network connections which use daemons started by *inetd*. By reviewing the connection log files, and looking at the patterns of connections and the time the connections occurred, it is possible to spot suspicious activity. All remote connections which are filtered using *tcpd* are

logged on the local host and well as the central security host. On the central security host, the *tcpd* connections are logged to the console window as well as a log file. We have a locally written suite of perl scripts which reduce the *tcpd* log files to a more readable form. See *figure 1* for a typical excerpt from one of the processed log files. The first line shows the name of the remote host which made the connection, and the number of connects (or attempts). The next lines show the time stamp of the attempt, the type of connection (e.g., *rlogin*) and the local host which received the connection.

```
[helmut.fim.wpafb.af.mil][3]
Feb15 08:48:43[rsh]           moon.nas.nasa.gov
Feb15 10:24:57[rlogin]        venus.nas.nasa.gov
Feb15 10:38:10[telnet]        moon.nas.nasa.gov

[hercles.dt.navy.mil][2]
Feb15 11:20:11[ftp]           moon.nas.nasa.gov
Feb15 11:20:29[ftp]           mars.arc.nasa.gov

[hertz.risc.rockwell.com][1]
Feb15 07:33:27[telnet]        moon.nas.nasa.gov

[hot.cray.com][2]
Feb15 07:03:30[ultra.ftp]     foobar.nas.nasa.gov
Feb15 08:23:23[ftp]           vee.unf.edu.fi
```

Figure 1: Excerpt from a *tcpd* connection log

There are several other programs available for host-based filtering. The *in.gate* program works in a similar manner as the *tcp_wrapper* software, in that it provides control over which hosts are allowed to use the services provided by *inetd*. The *in.gate* program also uses a separate configuration file like *tcp_wrapper*. There are also replacement programs for the *telnet*, *rlogin*, and *ftp* programs which filter connections as well. In a non-firewall environment, I highly recommend some form of filtering, either host-based or router-based, or a combination of the two.

Our next line of defense and monitoring is the extensive use of system logging information via *syslogd*. The *syslogd* program reads and forwards system messages to the appropriate log files as specified by the */etc/syslog.conf* file. The *syslog*d program has the ability to log many types of messages which range from kernel error messages to system daemons messages (e.g., messages from *ftpd*). Using *syslogd* to log important system messages is a crucial part of security. If system activities are not logged, then it is difficult to know what is happening or recognize unusual system behavior. For example, a log file which shows numerous failed login attempts to a single account might indicate someone trying to compromise the account. An audit trail of who became *root* would help trace actions back to a specific person. Log files provide an audit trail of system activity and are very useful in responding to security incidents.

At the NAS facility, we log as much information as possible. Authentication messages, such a failed login attempts and *su* commands, are logged.We also log all successful logins on the SGI systems, which provide this level of logging. Authentication messages for all NAS hosts are logged to the central security host, as well as the local host. On the central security host, we have a locally written program which reads the authentication logs and produces a summary report twice a day. On each NAS hosts, all general types of messages at the emergency level, all alert level messages and all mail related messages are logged to separate files on the local host. These log files are consulted only if needed. All system log files are archived either on a daily basis or a weekly basis, depending on the growth rate of the log file.

Important system messages, such as failed login attempts, *su*s to *root*, and *inetd* connections should be logged to a central logging host, as well as the local host. Intruders are known for removing or altering log files to hide their tracks. If important information is logged to a central host, loss of information less likely. Log files should be given consistent names and locations across all systems at a site to simply the task of locating similar types of logging information across multiple architectures. All system log file files should be

owned by system accounts and should not provide any type of access to normal users. At a minimum, three weeks of logging information should be kept. If the disk or tape resources are available, then increase the amount of archived data. From experience, I find that two or three months of logging information is beneficial.

Efficient and good account administration is also a crucial part of system security. Poorly installed accounts, accounts with poor or no passwords, and dormant accounts are still the most popular means for intruders to gain access to systems. At the NAS facility we have a locally developed central account management system, referred to as LAMS (Local Account Management System). All account activity (e.g., creation, deletion and modification) is done from a central host by automated tools which are a part of LAMS. The use of LAMS ensures that every account installed has an eight character, machine generated password, a home directory with default permission of 700, and a set of standard default environment files, with safe values for all common environment variables (e.g., *path* and *umask*). LAMS also provides the capability to change a user's password. This feature of LAMS has been used during several security incidents at the NAS facility. For example, if we suspect a NAS account has been compromised, we attempt to contact the user. If the user cannot be reached, we will disable the account and wait for the user to contact us. Once we hear from the user, the account will be re-enabled and the password will be changed using LAMS. The user is then notified and requested to change the password again.

The use of a centralized account management tool greatly reduces the time required to perform account operations. With the use of LAMS, accounts at NAS can be created or disabled on multiple hosts within a few minutes. The time factor becomes an important issue when a large number of accounts are involved in a security incident. There have been several cases at NAS where an operation was needed to be performed on several hundred accounts due to a security incident. Without the use of a centralized systems, such as LAMS, such a task would take many hours.

Monitoring account activity is another aspect of account management. We have a program which runs on a weekly basis to produce a report of all users who have not accessed their account for over 90 days. Due to the security risk of dormant accounts, all dormant accounts are disabled, and after an additional 30 days, the account is removed. If during that additional 30 days, the user contacts the accounts staff, the account will be re-enabled. When the account is disabled, the login shell of the account is set to a special program called *noshell*. The *noshell* program, which was written locally, reports on any attempt to gain access to the account (e.g., *rlogin*, *telnet*, *ftp*, *rsh*, *rcp*). Depending on how the program is configured at installation, a message can be sent to a monitor person, or a message can be logged via the *syslog* command. The message states which account was accessed, the time of access, the remote host (if the information is available), and the remote user (if the information is available). See *Figure 2* for an example message from the *noshell* program. There have been several occasions at the NAS facility where the use of the *noshell* program has revealed a security problem.

```
From: root
To: crabb@nas.nasa.gov
Subject: WARNING - Login to disabled account
**************** SECURITY ALERT ********************
"UNKNOWN REMOTE USER" has attempted to log into "badboy.nas.nasa.gov"
using the login name of "fredie"
Time of login: Wed Sep 18 14:46:22 1991
The remote login originated from: foobar.nas.nasa.gov
Internet Address of remote host: 129.160.33.145
Terminal line user attached to: ttyp11.
Please investigate this incident as soon as possible.
```

Figure 2: Example message sent by *the noshell* program

The final aspect of account management at the NAS facility, is monitoring changes to the */etc/passwd* file on all NAS hosts. This function is performed using a locally written program called *getall*. The *getall* program runs from a central location on a nightly basis. It copies down the */etc/passwd* file from each NAS host and compares it with the password file from the previous run. The program reports on additions and deletions to the file, changes in a user's login shell, changes to a user's UID/GID, or changes in *root* passwords. The *getall* program also sends a separate report on any new UID 0 accounts added to a password file.

One of the basic functions of good systems security is file system administration. File system administration includes such tasks as: ensuring system files are owned by system accounts, ensuring system files and directories are not group or world writable, ensuring critical system files are properly read protected, and the monitoring of important system files and directories for changes. At the NAS facility, we use several of the freely available security tools to perform basic file system auditing. The *COPS* package, which is one the most popular security checking tools on the Internet, is used at the NAS facility for the majority of our general file system administration. The *COPS* suite of tools is run on a weekly basis on all NAS hosts and the output reports are sent to a central person. All suspicious reports are investigated as soon as possible. Most of the problems reported by the *COPS* programs tend to be things that the system support people did (e.g., add a file to a system directory).

Auditing and tracking of SUID/SGID files can be a major headache at a site with a large number of hosts of multiple architectures. The *suid.chk* program, which is part of COPS, can be used to audit and track SUID/SGID programs. The *suid.chk* program does an exhaustive search of the file system and creates a list of all SUID/SGID programs. The list is then compared to a master configuration file and any differences are reported. The *suid.chk* program was modified at NAS to drastically reduce the work required to maintain master SUID/SGID files on numerous hosts. Instead of maintaining a separate master file on each host, the *suid.chk* program was modified to allow the use of group master files. There are currently 36 SUID/SGID master configuration files at NAS. A program called *get_group* is run by *suid.chk* to determine the master group for the local host. The *get_group* program also has an option to list all of the hosts of a specific group. This feature is handy if a file mode or permission needs to be updated on a group of hosts. The *suid.chk* program was also modified so the output report would show which master file was used to generate the report.

```
                Suid Audit Report for foobar.nas.nasa.gov
                =========================================
The Baseline file is: /usr/local/utils/cops/suid.files.sgi.group11

These files are newly setuid/setgid:

-rwsr-xr-x 1 root root 278640 Jan 18 16:13 /usr/sbin/xwsh
-rwsr-xr-x 1 root root 278640 Mar 30 1993 /usr/sbin/xwsh.badsum

These files are no longer setuid/setgid:

-rwsr-xr-x 1 root root 278640 Mar 30 1993 /usr/sbin/xwsh
```

Figure 3: Example report from suid.chk program

Generating the master files and determining which hosts belonged in which group was a very arduous and time consuming tasks. However, the weekly processing of reports and maintenance of the master files can now be done in several hours, whereas before it took many hours. The master configuration files, which are kept under RCS control, are updated on a weekly basis as the reports are read. Most of the changes required to the master files involve changing the file date stamp from a time stamp to a year stamp. Occasionally, a new SUID program will be updated or installed on all hosts, which requires the modification

of all master configuration files. The master configuration files, and the *get_group* and *suid.chk* programs are kept in the master source tree on a file server. Each week, the files are automatically updated on all hosts, via a cron job, prior to the run of the *suid.chk* program. See *Figure 3* for an example report from a run of *suid.chk*.

Another aspect of file system auditing involves the checking of permissions on users' home directories and their environment files (e.g., *.login*). One of the *COPS* routines, *home.chk* checks for permission and existence of users' home directories. However, in place of that program, we use *homecheck*. The *homecheck* program checks the permission and ownership modes of users' environment files as well as their home directories. The program also checks the permission and ownership modes of the parent directory of each home directory, and it can be configured to ignore certain accounts.

```
                    Rhosts File Audit Report For foobar
                    -----------------------------------

WARNING: Possible illegal or malformed .rhosts entries for user johndoe:
 sunny.larc.nasa.gov majdi

WARNING: Possible illegal or malformed .rhosts entries for user janedoe:
 uxh.cso.uiuc.edu aae391ac

WARNING: Possible illegal or malformed .rhosts entries for user jnsmith:
 rtccd.arc.nasa.gov *
```

Figure 4: Example output from the *raudit* program

The *.rhosts* and *.netrc* user environment files are audited using separate tools due to the security risk these files can create. The *raudit* program is used to audit users' *.rhosts* files. The program is run on a weekly basis to provide a list of all *.rhosts* entries which appear to be illegal (i.e., the login name in the entry does not match the account login name). The *raudit* program incorporates the use of an alternate login id database to reduce the report of false-positive illegal *.rhosts* entries. The *raudit* program is also used to audit all *.rhosts* files for the presence of specific hosts. This is done in cases where a security incident has been reported at a remote site where NAS users have accounts. For example, if I receive a report that systems at Berkeley were compromised and I know some NAS users have accounts on those systems, I will run the *raudit* program to look for *.rhosts* entries with *berkeley.edu* in the host field. The *raudit* program can also be used to produce a full report on *.rhosts* files with various statistics about them. For more information on auditing and managing *.rhosts* files, refer to the paper "Who's Trusting Whom? How to Audit and Manage Users' .rhosts Files" [1]. See *figure 4* for an example report from the *raudit* program.

The *netrc_chk* program, which is still under development, is used to report on users who have passwords in their *.netrc* files. Currently, the output from *.netrc* will show how many *.netrc* entries contain passwords for each user. The *netrc_chk* program ignores passwords that are of the form *username@host*. See *Figure 5* for example output from the *netrc_chk* program. Both the *raudit* and *netrc_chk* programs were developed at NAS and are available on request.

```
                    Netrc File Audit Report for Foobar
                    ----------------------------------

User johndoe has 3 passwords in his/her netrc file.
User janedoe has 1 passwords in his/her netrc file.
User jsmith has 5 passwords in his/her netrc file.
```

Figure 5: Example output from the *netrc_chk* program

Special access control and management on a large number of hosts can be a major problem, especially when a large number of people need the *root* password on multiple hosts. I define special access as the privilege to use one or more of the accounts necessary for support of the computer facility. The *root* account is one example. To reduce the difficulty associated with special access control in a large environment, there are several key tasks which should be performed. The first is the creation a database of users who have special access, why they need it and when their access expires. The database should also include all special access accounts and passwords. Systems of a like architecture (e.g., all SGI workstations) should share a common *root* password in order to reduce the need for several hundred different *root* passwords. A formal procedure for changing all special access passwords and distributing them should also be implemented. Special access passwords should be changed on a periodic basis (e.g., every three months or less).

At the NAS facility, we have 38 special access passwords (mostly *root* accounts) and over 100 people who require special access on one or more hosts. As a general rule, the special access passwords are changed every one to two months or whenever a person with special access leaves the project. The NAS special access password database is created and maintained on a Macintosh computer using the Hypercard application. Each user has a separate "card" in the database which lists the password groups the user has been approved to receive. Each time the passwords are changed, the user receives a new hardcopy password sheet. The password sheet contains the password for each group; however, the password printed on the sheet is not the actual password. The actual password is obtained by applying an algorithm to the password written on the sheet. The password algorithm is not written on the sheet and is the only item support personnel are required to remember. A full description of the NAS password management scheme can be found in the paper "Password Security In A Large Distributed Environment" [2].

Until recently, the special access passwords at the NAS facility were changed manually and the task usually took two people almost ten hours to complete. We began using LAMS to change passwords for all non-root special access accounts; however, it still took many hours to change the *root* password on over 300 systems. The task of changing *root* passwords was automated by the development of a small group of programs that use the client daemon from LAMS. The master driver program contains the hostname and password groups for all NAS hosts. The master program is used to create the lists of hosts for each password group, and the shell scripts which perform the actual password change. The password changing program works by modifying the password file to insert the new encrypted password string. The seven line program is shown in the *figure* 6. The use of this program requires the ability to write to the */etc/password* file and it requires every host to trust one host. LAMS was perfect for such a task, since it used a client-server daemon scheme to perform tasks, and trust was not an issue. The only manual task of a password change in the current scheme is to update the master program to include all new hosts, password classes, and the encrypted password strings. Now, all *root* passwords at the NAS facility can be changed in less than a hour!

```
#!/bin/sh
cp /etc/passwd /etc/passwd.bak

sed 's/^root:[^:]*/root:r9Xs5puJ9Obmo/g' /etc/passwd.bak > /etc/passwd

if [ !-s /etc/passwd ]
then
 cp /etc/passwd.bak /etc/passwd
 echo "Password Update Failed"
fi
```

Figure 6: Password file update script

Password validation, which is the process of ensuring passwords are well constructed and cannot be easily guessed by a cracking program, is an important security task because poor passwords are still one of the most popular methods used to gain access to a system. There are two types of password validation, pro-active and re-active. Pro-active password validation is the process of ensuring a password meets specific

construction rules at the time the password is set. Password construction rules include password length, and types of character. Re-active password validation is the process of verifying a password cannot be guessed by a password cracking program. Password cracking is less of a problem at sites that use a password shadowing mechanism. Many vendors now provide password shadowing as a standard part of the operating system; however, most flavors of UNIX still have weak password construction enforcement rules. Pro-active password validation requires the replacement of the */bin/passwd* program. There are three replacements available on the Internet: *password-plus*, *anl-password* and *npasswd*. All three of these programs operate in a similar manner. Each program has an external configuration file which specifies the password construction rules. Each program also allows the inclusion of a dictionary of words to check against when verifying a new password.

Re-active password validation is done using a password cracking program. In the last several years, there have been very large improvements in password cracking programs and several of them are freely available on the Internet. Several programs have even been written for the Connection Machine (CM-5). One of the more popular password cracking programs is *crack*. This program is available from CERT and several other sources. Matt Bishop also has a password cracking program which is available with his *Deszip* package. The Internet intruders have a very high-speed, widely used cracking program called *kc* (killer crack). They even have password cracking services available where you can email in a password file and they email back any passwords that were guessed by the program. Most of the password cracking services are operated by and used by the Internet intruders.

At the NAS facility, we are currently using both methods of password validation. We use *passwd-plus* as a */bin/passwd* replacement for all Sun and SGI systems. Currently, NAS passwords must be a minimum of six character long and must contain two of three groups of characters (letters, numbers and special characters). The */bin/passwd* replacement program is not installed on the Cray systems at this time because the Crays use password file shadowing, and the Cray version of */bin/passwd* has sufficient password construction enforcement rules. A recent policy was implemented at Ames Research Center regarding the construction of passwords. The new policy requires all passwords be eight characters in length and contain characters from three of the four groups (lower case, upper case, numbers, special characters). As a result of this new policy, we will be installing a new */bin/passwd* replacement program on all NAS hosts. Re-active password validation is done at the NAS facility using Matt Bishop's *Deszip* package. Re-active validation is usually only done when we suspect a NAS password file has been acquired by an intruder. The use of pro-active password validation has substantially improved the types of passwords used at the NAS facility. In the last three years, no passwords have been cracked during the re-active validation process.

There are a variety of other security tools freely available on the Internet aside from the tools mentioned here. There are several tools which can run for a single host to check the state of security on numerous hosts (e.g., a subnet of hosts). One of these tools is the *ISS* (Internet Security Scanner), written by Christopher Klaus, which can be used to scan a list of hosts for the presence of known security vulnerabilities or weaknesses in the system configuration (e.g., world exported file systems). An older program, called *sweep*, will scan a list of hosts and check for the vulnerabilities that were exploited by the Morris Worm. The *sweep* program was written at the Ballistic Research Lab in Maryland. The *SPI* (Security Profile Inspector), developed at Lawrence Livermore National Labs, performs similar functions to the *COPS* package. The list of security tools discussed here is by no means an exhaustive list of all the tools available.

Future Directions

There are still a number of improvements which can be made to the overall scheme of providing system security at the NAS facility. A number of projects are in the planning stage and several have been scheduled. One project involves the implementation of the *tripwire* program across all NAS hosts. The *tripwire* program is used to monitor a designated set of files and directories for any changes (e.g., unauthorized modification of files). Another project, which is currently in progress, is changing the systems default *umask* value. On most UNIX systems, the default *umask* set in the kernel is 0, which allows the creation of world-writable files. The user's *umask* value will be set at login time by reading a global environment file

such as */etc/cshrc* or */etc/profile* or is set in the *login* or *cshrc* program. However, this does not effect files which are created by system processes. Our plan is to set the system-wide default *umask* value to 077 by modifying the CMASK value in the kernel. Another project under consideration is the redesign of our network connection filtering scheme. Currently we are using a host-based scheme, via the *tcpd* program. We are interested in using a combination of router-based and host-based filtering to increase the effectiveness of filtering network connections.

Conclusion

Monitoring and managing security in a large heterogenous environment is not a simple task. However, there are a large number of security programs available which can make the task manageable and efficient. There is also a wealth of information on the Internet on how to maintain and improve the security of your site. The information is available for those willing to ask and to take the time to acquire the information. In this paper I have presented an overview of how the security is maintained the NAS facility, which can be viewed as a typical large scale heterogenous environment. While the methods and tools used to monitor and manage security at NAS may not be appropriate for every facility, they are a good starting place. As time and patience allow, new tools and procedures will be implemented at the NAS facility to further improve the security of the facility and decrease the time required to perform the work.

Availability

All of the programs discuss in this paper that were developed at the NAS facility, are freely available from NAS. To request the source for any one of the programs or policies discussed, send an email request to *doc-center@nas.nasa.gov*. If you have any questions regarding the use of the security programs discussed in this paper or how security is handled at the NAS facility, send email to *crabb@nas.nasa.gov*.

Author Information

Michele Crabb has been the primary computer security analyst for the NAS Facility at NASA Ames Research Center for over four years. During her nine years at Ames, Michele has worked in several divisions, in a variety of positions ranging from applications programming to UNIX system support. Prior to becoming the NAS security analyst, she was actively involved in providing system administration support for the large number of workstations at the NAS facility. Michele can be reached via electronic mail at *crabb@nas.nasa.gov*, or via US Mail at NASA Ames Research Center, Mail Stop 258-6, Moffett Field, CA 94035-1000.

References

1. Michele D. Crabb, "Who's Trusting Whom? How To Audit and Manage Users' .rhosts Files," *Proceedings of the Third Annual System Administration, Networking and Security Conference*, The Open Systems Conference Board, Apr. 1994.
2. Michele D. Crabb, "Password Security in a Large Distributed Environment," *Proceedings of the Second Workshop on Unix Security*, pp. 17-29, The Usenix Association, Aug. 1990.

APPENDEX A

Acceptable Use Statement for NAS Systems Division Computing Resources

The following document outlines guidelines for use of the computing systems, resources and facilities located at and/or operated by the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center. The purpose of these guidelines is increase awareness of computer security issues and to ensure that all NAS users (scientific users, support personnel and management) use the NAS Systems Division computing systems, resources and facilities in a efficient, ethical and lawful manner.

NAS accounts are to be used only for the purpose for which they are authorized and are not to be used for non-NAS related activities. Unauthorized use of a NAS account/system is in violation of Section 799, Title 18, U.S. Code, and constitutes theft and is punishable by law. Therefore, unauthorized use of NAS Systems Division computing system, resources s and facilities may constitute grounds for either civil or criminal prosecution.

In the text below, "users" refers to users of the NAS Systems Division computing systems, resources and facilities.

1. Users are responsible for using the NAS computing systems, resources and facilities in an efficient and effective manner.

2. The NAS Systems Division computing systems are unclassified systems. Therefore, classified information may not be processed, entered or stored on a NAS Systems Division computing system. Information is considered "classified" if it is Top Secret, Secret and/or Confidential information which requires safeguarding in the interest of National Security.

3. Users are responsible for protecting any information used and/or stored on/in their NAS accounts. Consult the NAS User Guide for guidelines on protecting your account and information using the standard system protection mechanisms.

4. Users are requested to report any weaknesses in NAS computer security, any incidents of possible misuse or violation of this agreement to the proper authorities by contacting NAS User Services or by sending electronic mail to *security@nas.nasa.gov*.

5. Users shall not attempt to access any data or programs contained on NAS systems for which they do not have authorization or explicit consent of the owner of the data/program, the NAS Division Chief or the NAS Data Processing Installation Computer Security Officer (DPI-CSO).

6. Users shall not divulge access information (e.g., Dialup or Dialback modem phone numbers, or lists of user accounts).

7. Users shall not share their NAS account(s) with anyone. This includes sharing the password to the account, providing access via an .rhost entry or other means of sharing.

8. Users shall not make unauthorized copies of copyrighted software, except as permitted by law or by the owner of the copyright.

9. Users shall not make copies of system configuration files (e.g. */etc/passwd*) for unauthorized personal use or to provide to other people/users for unauthorized uses.

10. Users shall not purposely engage in activities to: harass other users; degrade the performance of systems; deprive an authorized NAS user access to a NAS resource; obtain extra resources, beyond those allocated; circumvent NAS computer security measures or gain access to a NAS system for which proper authorization has not been given.

11. Electronic communication facilities (such as Email or Netnews) are for authorized government use only. Fraudulent, harassing or obscene messages and/or materials shall not be sent from, to or stored on NAS systems.

12. Users shall not down-load, install or run security programs or utilities which reveal weaknesses in the security of a system. For example, NAS users shall not run password cracking programs on NAS Systems Division computing systems.

Any noncompliance with these requirements will constitute a security violation and will be reported to the management of the NAS user and the NAS DPI-CSO and will result in short-term or permanent loss of access to NAS Systems Division computing systems. Serious violations may result in civil or criminal prosecution.

I have read and understand the Acceptable Use Statement for NAS Systems Division Computing Resources for use of the NAS computing facility and agree to abide by it.

Requestor's Signature: **Date:**

The Change Agent Mindset for Technology Infusion

Kris K. Bennett

Motorola - Cellular Infrastructure Group

About the Author

Kris K. Bennett is a Network and Systems manager within CIG's Information Technology Services organization. Kris is currently the project manager for the CIG Network Migration Project. She holds a Master's Degree in Computer Science from Illinois Institute of Technology and has over a decade of experience in supporting networks and computing systems. Kris can be reached via U.S. Mail at Motorola; 1501 W. Shure Drive; Arlington Heights, IL 60004 or electronically at: bennettk@cig.mot.com.

Abstract

For those of us sandwiched between our technical superstars and our executive management, acting as technology change agents within our respective companies is to say the least challenging. In seeking approval, funding and end-user support for a 2 year effort to migrate our current computing environment to a new architecture composed of X-terminals and various Unix servers, development of a change agent mindset was required to be successful. Approaches and strategies utilized to influence executive level management, as well as, end-user support to obtain multi-million dollar funding to migrate a software development community is discussed. In sharing experiences and approaches, it is hoped that others will be provided with ideas, suggestions and the motivation necessary to assist them in successfully infusing change into their respective organizations.

Introduction

Motorola's Cellular Infrastructure Group (CIG) is a multi-building campus setting where the infrastructure equipment to accommodate Cellular technology is designed and manufactured. In this campus setting engineers utilize predominately Unix workstations and servers to design, code, test and release hardware and software for the various products manufactured and sold. Currently activity to transition this community of engineers to a new computing environment is actively underway.

CIG Background

In the late 80's CIG expanded rapidly; engineering staff growth between 1986 and 1989 was tremendous. Deployment of SUN 6800 based servers and workstations to accommodate that growth began with 40 nodes and expanded to more than 1200. To support such explosive growth, the CIG Network and Systems support group also grew from approximately 5 to a staff of over 20.

In late 1990 the growth in the engineering hiring stablized and flattened. At the same time, some of the workstations deployed in the early phases of the 86-89 growth period were now 4 years. In addition, the network was now integrated into our engineering development processes. Network reliability and system availability began to play an increased role in CIG's ability to deliver switching products on time.

Between 1989 and 1993, other factors influenced general change through the company and with respect to the computing resources utilized by the engineering development community. Our network backbone was transitioned to fiber and Cisco routers were purchased and installed. SUN announced their departure from the use and manufacture of their Motorola 68020 and 68030 line of products. The impact of

their announcement effected the CIG Network and Systems support group with respect to driving our focus towards the use of products utilizing Motorola chip sets. As a result, expansion in server resources was accomplished utilizing Auspex file servers. Between 1990 and 1992, a total of 10 servers were purchased and installed. With respect to workstations, the use of NeXT computers was investigated during this timeframe as well, but no movement with respect to transitioning out some of the older SUN workstations ever occurred. As a result, engineering development and testing continued to rely on the 2-3 MIP workstations being utilized to support our engineering efforts.

In 1991 new executive level management entered CIG and a focused message to move our cellular products from proprietary systems to open systems designs was repeatedly conveyed throughout the organization. In addition to our new management, corporate executives were sending messages to reduce cycle time by a factor of 10 and continue to improve overall product quality. In 1992, a major push towards these initiatives was underway. Another message that was being sent was to utilize small, focused teams as a means of driving needed changes. Participation in Motorola's Total Customer Satisfaction (TCS) competition was highly encouraged within CIG as a means to form small teams and accomplish needed changes to reduce cycle time, reduce costs and improve quality. As a result of these frequent and consistent messages, an observable transition in thinking began to take hold throughout the company and as a result, our business conditions began to improve as well.

For the network and it's supported engineering community, the transition to a new way of thinking and doing business over shadowed the problems and performance issues with the network for the first year under our new executive management. But as our engineering business units transitioned from that of an ad-hoc operation to one focused on developing common processes and practices, the importance of a reliable, stable and performing development environment began to uncover the problematic and aging network environment as a item needing attention. Although the Network and Systems group was acutely aware of this fact and had been for some time, it wasn't until it became apparent to the network consumers that the network and computing environment did not meet the demands of our new business initiatives and practices. These events essentially set the stage for our current transition activity.

Challenges Faced

Although general agreement regarding a need to transition our current computing environment existed, the funding and effort to do so would be significant and justification would be required. The severity of the situation and the importance of the computing environment in supporting CIG development activities would have to be clearly demonstrated and some method of projecting the benefits of a major migration specifically outlined.

Estimating the effort and outlining the deployment activities necessary to migrate the entire engineering community was relatively easy. Lessons learned during the 86-89 growth period at CIG provided a historical base from which to draw conclusions regarding the staffing needs, the development activities required and the deployment rates which could be achieved. Based on that historical experience and the current size of the engineering community, a migration from the current computing environment was estimated as a 2 year effort. In 1992, a general framework to migrate our computing environment was developed and a target start date of January '93 was established.

Although effort estimation and a general project plan existed, demonstrating the importance of such an extensive change and the benefits to be achieved was a major challenge to be faced. In the fall of '92 during an all day management meeting, an unknown opportunity arose to address this particular challenge. A discussion regarding the development and execution of an employee environment survey to be sponsored by a divisional Vice-President along with suggestions to automate the survey provided us an opportunity to support a management initiative. As a result of volunteering to automate the survey, our involvement expanded from simply automating the survey to participating in the generation of the survey questions and the analysis of the results. Although no questions directly assessed the computing

environment, comment results identified the current computing environment as the largest, single negative factor effecting the engineering environment. In January of '93, we presented the survey results at a similar management meeting, and the data exposed our organization to jeers and sneers but ironically made this problem acutely visible to an entire division management and initiated momentum for change.

Our group had been involved in assessing the feasibility of utilizing a server/X-terminal environment for engineering use throughout '92 and was gearing up to initiate deployment to a beta test group. Deployment occurred in March of '93 and was timely given the resent results of the employee environment survey and the obvious need to address our current computing environment issues. This was a vital turning point in our migration activities and initiatives and introduced some interesting challenges and insights for the team.

Approximately 7 months of activity had taken place in gearing up for our beta deployment. Work in overviewing the application needs and a strategy for successfully integrating the new platform into the current network had been constructed to ensure a smooth and effective transition. Unfortunately, problems resulted that were significant enough to cease further expansion into the engineering community at that time.

Although numerous problems were encountered, working within the framework of servers with X-terminals for engineers began to demonstrate some interesting advantages in using what we now refer to as an "X-architecture". The "X-architecture" demonstrated an open and flexible environment which could be adjusted quickly to meet the changing needs of a business and the supported users. Network wide upgrades could be addressed more quickly simply due to the fact that it alleviated the need to upgrade the workstations residing on engineers desks. It therefore reduced the potential of the network support staff to become a bottleneck for addressing OS upgrades, bug fix installations or other tasks required by the engineering users. The architecture also demonstrated the ability to support multiple platforms and applications and could support their introduction and availability to network users with minimal disruption. The X-terminal was viewed to actually give more control back to the end-users by providing them with some redundancy in available servers and by providing them the ability to recover in the event of server failures without the intervention of the support group. In addition, the costs to implement and maintain this architecture were significantly less.

The effort in pursuing the X-architecture would be significant on a large scale. Despite its advantages, the architecture would dictate a redesign of the entire current network layout. Understanding the differences in display traffic and inter-server traffic would be necessary if we were to proceed with the "X-architecture" strategy throughout CIG. In addition, the effort to migrate existing applications and software to any new computing environment needed to be researched and the effects and effort on and of the user population understood. Although, some of these issues had been addressed on a small scale with our beta deployment in March of '93, the magnitude and effort of wide-scale implementation was another story; and to top off these complexities, resistance to a non-workstation strategy was strong among the engineering community.

The resistance was not limited to the engineering end-users, however. Within our own team, views were mixed. The complexities and issues to be overcome in proceeding with the "X-architecture" strategy were thought by some to be enormous. In addition, they were thought to impact our ability to delivery a solution to our end-users in a timely fashion. Time was critical and a delay in introducing a solution that would meet the engineering needs was viewed as risky. Essentially, implementation of a workstation based environment with faster machines and new applications that could support the engineering activities was viewed as a safe, fast an effective means of delivering a solution. And this is where the need for a change agent mindset enters.

Establishing the Mindset

As technologists, the benefits of new technology often seem obvious to us however, the advantages of a technology itself is not enough to get the green light and funding required to proceed with introducing a technology change. One has to equate the technology to a business benefit and in addition align it with business goals and objectives. In establishing the change agent mindset, one needs to develop or tune their awareness, involvement and understanding of fundamental business goals and objectives. The technology can then be infused into the company rather than simply injected becoming a supportive and necessary element in driving change. It then becomes an enabling solution helping to meet business initiatives.

It is part of our responsibility to be aware of the changing technologies in our profession; however if the introduction of a particular technology doesn't benefit our business goals and objectives then it's probably not the right technology. Even if it is from our technologist perspective, it's likely that support and funding from executive level management may be difficult. Thus, each situation and opportunity we confront with respect to driving change must be measured against its capability of benefiting our businesses. Once assured that the benefit and opportunity to supplement and support those initiatives is demonstrated, strategies must be formulated to bring the change about.

So if we reexamine the migration efforts at CIG with a change agent mindset, one will see that the X-architecture is open and flexible, how it has the potential to save money, has the opportunity to improve productivity, reduce cycle time and in a general sense how it aligns with the executive level management as well as corporate management driven goals, initiatives and objectives.

The real problem we were faced with was that the server platform used in our beta deployment activity was not adequately meeting the needs of the engineering community. A solution was required that could meet their needs and a strategy needed to be developed to bring the change about. A strategy was then developed to do just that. A slight variation in our current work with the X-architecture was thought to be an appropriate solution and enabling strategy such that we could meet the needs of the engineering community and support the business objectives. In addition, the twist that was introduced was able to calm some of the fears within our own group regarding our ability to introduce a solution quickly.

The Project Roadmap

Activities Up Through Approval

Activity began on analyzing the problem from a business perspective. Essentially, a proposal justifying the proposed architecture was written. The proposal outlined the benefits of the X-architecture in terms of implementation and maintenance cost savings, flexibility, security, and in general terms, cycle time gains that would be achieved once implemented. A series of meetings were conducted to gather bids from alternate vendors to compare the costs of an X-implementation to that of a workstation implementation. In conjunction with this research, work was initiated within our group to test the X-architecture with other platforms. Two months of intensive work was conducted in preparation for requesting the needed funding to initiate our migration project.

In May of '93, following some of this initial feasibility work, factual information regarding the problems that had been experienced with our March beta deployment was presented to General Management one week before presenting our request for funding implementation of the X-architecture with an alternate platform. The point here is that we sincerely worked to integrate the initial platform we had been using, we documented our findings and the problems and we constructed a back-up strategy and solution that we felt would be able to meet the needs of the engineering community, the initiatives of our management, and benefit our business.

When presented with the strategy, management saw the benefits of this strategy as well. The architecture was "open" and "flexible" - essentially it aligned with the message that CIG executive management began preaching when they arrived in '91. However, despite buy-in to this approach by the team and executive level management, a number of engineers still insisted that despite some of the perceived advantages of the X-architecture, it would not provide them with the performance that they required and they were resistant to the idea of sharing resources. In order to succeed, we needed not only management buy-in but the engineering community as well. In addition, we had to be sure this was the right approach. Otherwise we would be setting ourselves up for a support nightmare.

To gain engineering support it would be necessary to demonstrate the advantages of the architecture as well as demonstrate that concerns of the engineers could be addressed. Questions as to what other companies were using an X-architecture had to be anticipated. Concerns regarding performance and resource sharing also had to have solutions. Information was sought outside the company. Various vendors were asked about their thoughts and perceptions. Information was gathered from other sites that had taken this approach. In our efforts to identify other sources, we uncovered another division with Motorola with a similar mix of platforms and a similar user community profile. They had just implemented a similar transition in their business. Their results were good, but they had no quantifiable data to justify the benefits yet as they had just completed an initial phase of their implementation. In addition, their customer base was significantly small relative to ours.

We performed some preliminary testing of the architecture and invited representatives from our user community to come and see what was being proposed. This activity was performed prior to our request for funding. It was essential to have some level of buy-in from the engineering community before seeking funding. Prior to these gatherings, we had developed answers or strategies to address the set of anticipated questions but in addition, this setting provided us with an opportunity to get input and feedback directly and to also gain a feel for the extent of the opposition. These sessions also were beneficial in pointing out some areas that we had not anticipated to be problematic from the view of the engineers and gave us an opportunity to initiate investigation into solutions for these new concerns.

To summarize the activities thus far, we had conducted a beta deployment in March and following problems with that deployment began work on researching alternative platforms. During our research and verification of those alternative platforms, formal proposals from vendors were initiated. Funding was formally requested in May of '93 following our beta deployment results presentation, our engineering feedback sessions on the architecture and near the tail-end of our bidding process. Platform selection was made following the approval in funding and based on approximately 1 month of platform evaluation research.

Activity Following Approval

With the approval of year 1 funding for our 2 year migration project, we initiated work in a number of areas to prepare for implementation. Our group met to determine what had to be performed in order to implement this migration tasks within two years. During brainstorming sessions, we categorized the activities and identified groups of individuals within our group to champion the various efforts. Essentially we identified the work in terms of these major areas - Project Planning and Management, Technical Design, Network Design, End-User Interface, and Implementation. Each of these major activity headings included a number of sub-activities and tasks that would have to be performed in order to successfully deploy our new computing environment.

Immediate work began on purchasing systems and initiating our development integration work. This activity focused predominately on Operating System and Network services issues such as drive partitioning, filesystem layout, kernel configuration, NIS, DNS, etc. A portion of this work included "fixing the evil" in the current environment and attempting to alleviate items which repeatedly caused problems for our end-users and often resulted in them calling in the support staff for resolution. We also took this opportunity to implement enhanced security features.

While equipment purchases, development and integration work was underway, activity on a larger scale was initiated within our engineering community to overview the migration project. An initial meeting to engineering level management was conducted, and project group representatives were identified to work with us in integrating this new computing environment. Once again, our previous experience in large scale deployment back in the late 80's was called upon to assist in constructing a process which could be used to outline the steps necessary to control and manage the deployment activity. As part of that process, a matrix of groups, their corresponding champions and the various activities that had to be stepped through prior to a group's conversion were outlined. The matrix served as a tool for managing the conversion of groups to the new environment and additionally, it served as a framework to monitor deployment progress. The initial step of our deployment matrix consisted of working with the designated group champion to determine the requirements of that group. In addition, it was necessary to determine whether or not a group was dependent upon others in terms of software tools, documentation or data sharing dependencies for which we might not be aware. These requirements and dependencies were then used to map the deployment schedule. The intent of the matrix was to control deployment, monitor progress and ensure that upon conversion to the new environment, disruption in current engineering activity was minimized.

Network design activities were tied closely to our integration and development work. Preliminary testing was conducted using prototype systems and a test LAN. Our Implementation champion worked on hardware configuration and computer room facilitization issues in conjunction with our Network Design and Technical design champions. Software requirements were fed from the End-User Interface champion into our Technical design team so that a standardized software environment could be developed and so that a standardized software distribution mechanism could be designed.

A beta deployment group was identified and a deployment target date established for September of '93. The group was composed of approximately 60 engineers. We attended a departmental meeting and overviewed the architecture and environment to be delivered to the beta group. This meeting also provided our beta users an opportunity to ask questions regarding the new computing environment we were about to deliver to them. During this meeting, we announced the availability of a porting environment and released it to this targeted group 1 month prior to their cut-over date so that they could initiate porting efforts that we could not address for them. A note should be made here that the selected beta group was identified as group supportive of the proposed migration strategy. In addition, they were using a technology that we felt should be adopted more widely throughout the organization and therefore, in working closely with them, we were provided an opportunity to learn from them as well. In addition, this group was interacting fairly extensively with another engineering group who was still significantly resistant to this proposed architecture even though we had been given a green light. And finally, their dependency on other groups throughout CIG was minimal.

The beta test duration was 1 month from cut-over and during that time, we tracked problems, adjusted our network design, and tuned our processes and procedures which had been just used to accomplish the beta deployment. In addition, to these adjustments, a dedicated migration team was established improving our alignment with executive level goals to utilize small focused teams. In doing so, we had an individual dedicated 100% to some aspect of the migration project and we were able to intensify our deployment activity. With this regrouping, we also introduced an additional champion to address migration software issues. This adjustment was made because following our recent beta deployment, it was evident that additional effort was required to address all of the software issues related to migrating a particular group.

After fine-tuning our process, procedures and team, we keyed on migrating groups who develop and maintain tools utilized by other engineering groups. These "tools" groups were targeted because other groups were dependent on the availability of their tools and since one of our goals was to deliver a more productive environment, we couldn't afford migrating a group before the tools they required were in place.

Helpful Hints

This can be an area of caution as well. Understand and keep in touch with these "tools" groups. They often have innovation solutions to software application problems, but sometimes their understanding of the network and how to design applications that play well with a very distributed environment or large scale network is not quite as good as it may need to be. Most of us have been in the position at one time or another of solving software design problems by throwing hardware at the problem. This is especially true for applications that do not account for or take advantage of a distributed environment. In any event, watch these application and tool development groups. Keep abreast and get involved with their project activities to assist in an effective deployment of large scale applications.

During these initial deployments, opportunities arose to seed the technology into pockets of the engineering community that were still resistant to the proposed architecture. One failing was that we introduced a group to the technology and architecture to meet an immediate business need but in doing so we deviated greatly from our guidelines and processes. The result was less than successful. In another case, we migrated a group that was physically next to a skeptical group. Up until that point, the designated champion of the skeptical organization had been less than cooperative about working with us to define requirements. However, following our migration of a nearby organization, the uncooperative champion was knocking down our doors to understand the project and asking what needed to be done to get on the deployment schedule. The lesson learned in regards to seeding was that seeding can be advantageous to introducing the technology to groups skeptical of the technology by teasing them essentially. But don't seed just to seed, be sure to stick to implementation guidelines, policies and processes and don't take risks which may increase the level of opposition.

Informal communication may also aid in swaying opposition; however, banking on the grapevine is probably not a good idea. However, realize that this can occur and don't be surprised if it does. If the infusion of a technology is successful in one segment of the company, groups initially opposed to it may begin asking more questions and wanting to get more involved in understanding the project and what it might be able to do for them. If this happens, know that progress is being made in respect to driving change and impacting the business positively.

Other aspects of our current project that have been important in our success to date deal with communication, training and team formation. As part of our migration project a training course has been developed to ease the transition for the engineers to the new computing environment. As part of this effort, time was spent generating a user transition manual and a 1 day course to review the architecture and to equip the users with a working environment the week of their conversion. In addition, we have written procedures and have held a 1 day training session to educate our own support staff on all the goals and objectives of the project, the project timeline and the technical and system administrative changes implemented. In formalizing these materials, we produced binders and manuals which exhibit our project logo. The logo attempts to summarize some of the fundamental initiatives of our project. Although a small contribution given the scope of this project, it has been beneficial in communicating our message and in keeping our project visible throughout the campus.

In respect to the team of individuals driving this transition and migration effort throughout CIG, the importance of having the right individuals championing the efforts on a project of this magnitude have been crucial. Their motivation, knowledge, drive and hard work are very visible and recognizable to our user community. Although their skills and knowledge differ, they complement one another. In addition, many of them were previously in competition with one another as they were performing similar functions within the organization. In providing them a common goal by way of this project they have been able to excel in terms of their individual contributions. This has benefited the project greatly and has helped them to accomplish more than many ever expected.

Current Project Status

At this stage in our project we are converting approximately 100 engineering seats a month over to the new computing environment and we are nearing our half-way mark in terms of the number of engineers to be migrated. We are currently in our 6th month of deployment and still within our first year since receiving funding. We have a fine tuned process and effective procedures in place. Our efforts thus far have been applauded by our users and the benefits of this project are beginning to demonstrate the impact this change has made within CIG. Cycle time improvements in some areas of our engineering development activities are nearing our corporate 10x reduction goals. In addition, we have also had assistance from various engineers in respect to migrating themselves to the new computing environment. Each of the product group champions has expended effort to ensure their group's needs are met and other engineers have assisted in deinstalling their old workstations and setting up their new X-terminals on the night of cut-overs. This project has truly demonstrated the power of teams and the significance of working together towards a common goal. Most importantly, we are establishing a solid computing foundation capable of improving our business effectiveness today and flexible enough to support our future initiatives.

Strategies to Consider

The change agent mindset is not something that most of us have. It is something that requires development. Often it is developed through experiences and lessons and by identifying common threads in successful endeavors. However, an awareness of business objectives and goals and some common strategies can assist in driving change and formulating enabling strategies and solutions. Some of those strategies and techniques can be summarized as follows:

"Don't Leave Home Without It"

- Review a the technology from the standpoint of what it will do for the business and how it maps to the initiatives and the goals of the executive management.
- Keep in mind that customers are your greatest asset in helping to drive the change into place. They represent the business and their support is critical in introducing change successfully. Take steps to get them involved and keep them involved.
- Gather data and use it to solidify a foundation for change. Conduct surveys or participate in any activity that can provide insight into the situation from the customer's perspective.

"Look Outside Your Walls"

- Look *outside* the confines of your company and make sure that what sounds like a good idea is one. Identify other companies or sites that have successfully implemented something similar. Make sure some level of success exists and use whatever insights or experience those examples illustrate to assist in avoiding potential pitfalls.
- In implementing a change, consider whether or not this is a good time to address unrelated problem areas. Make a conscious choice whether or not to address them during implementation. Consider their effect on the overall project plan and schedule and again checkpoint them against the question - "what will this do for our business?" Take into consideration activity *outside* the core project scope as a opportunity to institute needed changes.

"Make Problems Visible"

- Make problems visible. If they are visible, then you're more likely to be afforded the opportunity to take a new direction or approach to resolve them.
- Develop mechanisms and processes to track and monitor project progress and to also make problems visible.

"Communicate with Coins"

- Coin phrases or use logos to help communicate the goals and objectives of the change to staff as well as end-users.
- Make sure to justify the change in terms of dollars and cents. There must be a payoff for the change.

"Tease Your Adversaries"

- Seed groups with the technology. Introduce it initially into a supportive group, then seed it near groups skeptical about the technology. Tease them with the technology.

"Work the Network"

- Consider the potential that informal communication may have in assisting a change of view/attitude in a camp uncertain and concerned about the proposed technology.

"Watch your Neighbors"

- Keep mindful of the activities and projects being driven by the application and software tools organizations. Participate in their development activities when possible to aid in their successful implementation of large scale applications.

"Think Team"

- Make conscious decisions in forming the right team to implement the technology and drive the change.
- Integrate customers as part of the technology team.

Phases were associated with the above strategies to help in communicating their messages. Just as a logo was used to communicate the fundamental goals and initiatives of the CIG migration project, so too are these phrases. It is hoped that they will assist in recalling a set of strategies that may be useful in a time when a change agent mindset is needed.

Conclusion

Essentially a case study has been presented. Much of the information focused on mapping the activities that transpired to successfully obtain the needed funding and end-user support to induce a significant change in the computing environment at Motorola CIG. By outlining those activities, various lessons were learned and various techniques were implemented. It is hoped that some of the techniques and strategies used here will be beneficial to those faced with similar challenges and that by considering

them or through their application, technology can be successfully infused to support goals and initiatives.

As change agents, our efforts must be continuous. A constant watch on potential opportunities in which technology can be used to support business initiatives must be maintained. In addition, the development and exercising of a change agent mindset must be continuos to bring needed change about.

Acknowledgements

I'd like to acknowledge Dave Opferman, Vice-President and General Manager of CIG's Cellular Systems Division and Jack Leifel, Director of CIG's Information Technology Services (ITS) for their continuos inspiration, support and guidance in implementing this change throughout CIG. In addition, the ITS Network Migration Team members need to be recognized for their long hours of dedication, effort and support in getting us to where we are today.

References

[1] Bennett, Kris K., *CIG Network Migration Project Plan*, Internal document of Cellular Infrastructure Group, Motorola CIG, 1993.
[2] Buchanan, Leigh, *Going for the Gold*, CIO, Volume 7, Number 7, January 15, 1994.
[3] Karten, Naomi, *Mind Your Business*, QED Information Sciences, Inc., 1990.
[4] Lundy, James L., *TEAMS*, Dartnell Press, 1992.
[5] Ould, Andrew, *How Motorola Lost the Workstation Market*, UnixWorld, July, 1990.

ISBN 1-880446-59-6